



Hands-on Lab

Session 9402

IBM Integration Bus

KafkaProducer node ... OR ...
LoopBackRequest node ... OR ...
A choice from lots of other options!

Provided by IBM BetaWorks

© Copyright IBM Corporation 2017

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

This document is current as of the initial date of publication and may be changed by IBM at any time.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

Table of Contents

1. OVERALL LAB DESCRIPTION.....	5
1.1 WHAT DOES THIS LAB COVER?	5
2. KAFKA LAB: INTRODUCTION AND PREPARATION	6
2.1 INTRODUCTION	6
2.2 SCENARIO.....	6
2.3 KAFKA SERVERS	6
2.3.1 <i>Model Definitions</i>	7
2.3.2 <i>The HR_Service REST API</i>	8
2.4 CONFIGURE TESTNODE_IIBUSER FOR REST APIs	9
2.5 CONFIGURE INTEGRATION BUS NODE TO WORK WITH DB2.....	10
2.5.1 <i>Create database and tables</i>	10
2.5.2 <i>Create JDBC and security configurable services</i>	10
2.6 OPEN THE WINDOWS LOG MONITOR FOR IIB	11
3. EXPLORE AND START THE KAFKA SERVERS.....	12
3.1 KAFKA CONFIGURATION FOR IIB WORKSHOP	12
3.1.1 <i>Kafka installation notes</i>	12
3.2 EXPLORE THE KAFKA CONFIGURATION	13
3.3 START THE KAFKA SERVERS.....	14
4. IMPORT AND EXTEND THE HR_SERVICE REST API.....	18
4.1 INVESTIGATE THE CREATEEMPLOYEEMAIN SUBFLOW	24
5. TEST THE UPDATED HR_SERVICE REST API.....	30
5.1 USING SOAPUI.....	30
5.2 USING POSTMAN.....	35
6. USING THE KAFKA NODES WITH IBM MESSAGEHUB.....	37
6.1 EXPLORE AND CONFIGURE MESSAGEHUB.....	37
6.2 TEST HR_SERVICE WITH MESSAGEHUB.....	43
END OF KAFKA LAB SCENARIO	44
7. LOOPBACK LAB: INTRODUCTION AND PREPARATION.....	45
7.1 INTRODUCTION	45
7.2 OPEN THE WINDOWS LOG MONITOR FOR IIB	45
7.3 CONFIGURE TESTNODE_IIBUSER FOR REST APIs	46
8. CREATE THE HRDB DATABASE IN MONGODB.....	47
8.1 START THE MONGODB SERVER AND CLIENT SHELL	47
8.2 CREATE THE HRDB DATABASE.....	49

9. CONFIGURE THE IIB INSTALLATION.....	51
9.1 INSTALL THE LOOPBACK CONNECTOR	51
9.2 CONFIGURE DATA SOURCE FOR LOOPBACK CONNECTOR	52
10. CREATE AN IIB REST API TO ACCESS MONGODB.....	53
10.1 CREATE THE NEW REST API	53
10.2 ALTERNATIVE 1: SET LOCALEENVIRONMENT "FILTERSTRING" USING MAPPING NODE	61
10.3 ALTERNATIVE 2: SET LOCALEENVIRONMENT "FILTERSTRING" USING ESQL COMPUTE NODE	69
10.4 COMPLETE THE FLOW DEVELOPMENT	71
10.6 DEPLOY AND TEST THE REST API	74
END OF LOOPBACK LAB SCENARIO	76

1. Overall Lab Description

1.1 What does this Lab cover?

This is an Open Lab which has been timed for one hour.

This set of printed hard-copy instructions provide you with a choice.

Select **ONE OF** either:

- **KAFKA LAB:** The KafkaProducer node was added to IBM Integration Bus in version 10.0.0.7. In this lab, we use an IIB REST API containing a KafkaProducer node to write a message to a local Kafka implementation. We then use the KafkaProducer to send data to the Bluemix Message Hub service.
- **LOOPBACK LAB:** The LoopBackRequest node was added to IBM Integration Bus in version 10.0.0.6. In this lab, we create an IIB REST API which exposes MongoDB. The IIB LoopBack Request node is used in the message flow to retrieve documents from the database.
- In the unlikely event that you don't want either of these options, we would also like to offer you the opportunity of using this Lab's VMWare image in conjunction with a wider choice of (32!) possible Labs using soft-copy instructions which can be downloaded in PDF format from this URL: <https://ibm.biz/betaworks-iib>

Communities

BetaWorks - IBM Integration Bus

IBM Early Programs

▼ **BetaWorks - IBM Integration Bus**



Overview

Recent Updates

Members

Files

Forums

Bookmarks

▼ Tags

Find a Tag

1605 add cloud connectors

endpoint iib images mq

on please questions rest

salesforce this to topic with

You are in Community: IBM Early Programs > BetaWorks - IBM Integration Bus

▼ Community Description

Welcome to the IBM BetaWorks IBM Integration Bus (IIB) community. The IBM BetaWorks team are using this community to distribute the hands on instructional Lab Guides that we prepare for our beta workshops. If we decide to publish a Lab Guide in this Community the feature it promotes will be available in the latest level of IBM Integration Bus. The Lab Guides are made available here on an "as is" basis to help you experience the latest features of IBM Integration Bus.

Use the Files section (see the link in the left nav bar) to see the latest Lab guides.

The Lab Guides are provided on an "as is" basis however, if you have a question on a lab guide, please use the pinned topics on the Forum (link on the left of this page) and we will try to help. Note you will need to "Join this community" in order to post to the Forum.

If you need to download a Developer version of IBM Integration Bus please follow this link

A note about the Lab Guides: The Labs have been written on a Windows based IIB Environment. Some Lab guides use software that will require installation in your environment, unfortunately we can only answer questions relating to IBM Integration Bus.

The IIB IBM BetaWorks Team (November 2016)

Files

Files Folders

 IIB student folder (Latest Version 20th January 2017)	Updated on October 5, 2016 by Terry Hudson	
 IIB Hands on Labs (IIB 10.0.0.7)	Updated on 1/18/17 by David Hardcastle	
 IIB Hands on Labs (IIB 10.0.0.6)	Updated on December 12, 2016 by David Hardcastle	

2. KAFKA LAB: Introduction and Preparation

2.1 Introduction

The Kafka Producer and Consumer nodes were introduced in IIB v10.0.0.7. In this lab you will use the Kafka Producer node.

2.2 Scenario

The scenario described in this lab uses an IIB REST API which updates a database HRDB. Updates that are made to HRDB will also be propagated to the Kafka server. The lab will start from a partially-built solution of the REST API called HR_Service which we will then extend to communicate with Kafka.

2.3 Kafka servers

This lab will investigate two Kafka servers:

- We will begin by investigating the use of locally-installed Kafka servers. Development and testing of this lab was done with three Kafka servers installed locally on the IIB workshop Windows system. These three servers have configured Kafka replication (replication factor three can be used for topics)
- Next we will reproduce the same scenario, but using the IBM Bluemix MessageHub service instead of the local Kafka servers. MessageHub is IBMs implementation of Kafka on Bluemix.

2.3.1 Model Definitions

The following message models are used by this updated version of the HR_Service REST API.

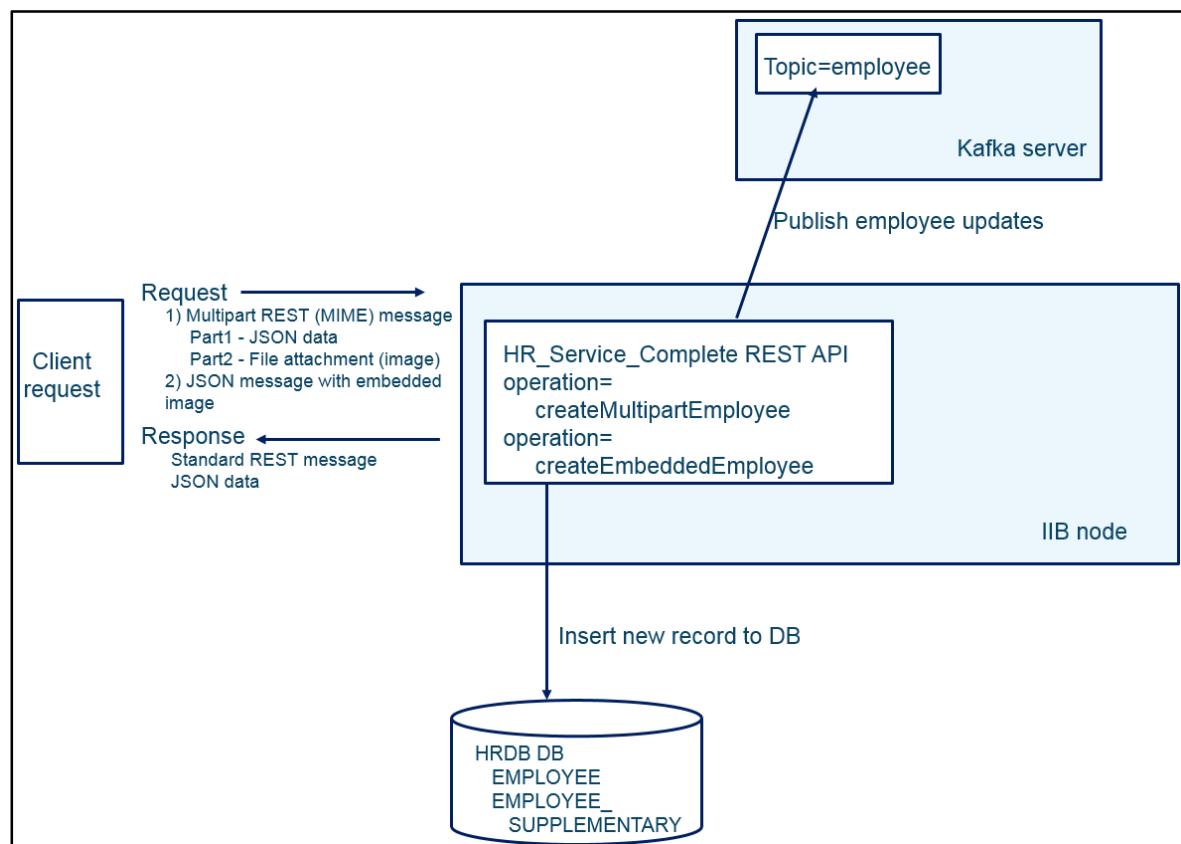
- DBRESP – contains database response information
- EMPLOYEE – defines columns in the EMPLOYEE table
- EMPLOYEE_SUPPLEMENTARY – defines columns in the EMPLOYEE_SUPPLEMENTARY table
- DEPARTMENT – defines columns in the DEPARTMENT table
- EmployeeResponse
 - DBResp (type = DBRESP)
 - Employee (Array, type = EMPLOYEE)
- DepartmentResponse
 - DBResp (type = DBRESP)
 - Department (Array, type = DEPARTMENT)
- CompleteResponse
 - DBResp_employee (type = DBRESP)
 - Employee (type = EMPLOYEE, single object, not array)
 - DBResp_department (type = DBRESP)
 - Department (type = DEPARTMENT, single object, not array)
 - DBResp_employee_supplementary (type = DBRESP)
 - Employee_supplementary (type = EMPLOYEE_SUPPLEMENTARY)
- EmployeeSupplementaryResponse used when only accessing EMPLOYEE_SUPPLEMENTARY
 - DBResp
 - Employee_supplementary
- EmployeeAddUpdateCompleteRequest (input message, used when adding a complete new employee)
 - Employee
 - EmployeeSupplementary

2.3.2 The HR_Service REST API

The partially-built solution version of the HR_Service REST API has implemented six operations:

- getEmployee
- getDepartment
- getSupplementary (retrieves data from the EMPLOYEE_SUPPLEMENTARY table)
- getComplete (invokes the getEmployee, getDepartment and getSupplementary operations)
- createEmployeeFromMultipart (adds new employee using a MIME request)
- createEmployeeFromEmbeddedImage (adds a new employee with a plain JSON request)

The four “get” operations are not used in this lab scenario. The two “create” operations invoke a common IIB subflow that performs database inserts into the EMPLOYEE and EMPLOYEE_SUPPLEMENTARY tables. In this lab, you will extend this subflow to publish a notification message to the Kafka server when a new employee is added to the EMPLOYEE tables.



2.4 Configure TESTNODE_iibuser for REST APIs

The instructions in this lab guide are based on a Windows implementation, with a user named "iibuser". If using the workshop VMWare system, login to Windows as the user "iibuser", password = "passw0rd". (You may already be logged in). Start the IIB Toolkit from the Start menu.

The IIB support for the REST API requires some special configuration for the IIB node and server. Cross-Origin Resource Scripting (CORS) must be enabled for the IIB node to execute REST applications. This is also required when testing with the SwaggerUI test tool. See http://www.w3.org/TR/cors/?cm_mc_uid=09173639950214518562833&cm_mc_sid_50200000=1452177651 for further information.

1. Ensure that TESTNODE_iibuser is started.
2. Check that CORS has been enabled on the IIB node by running the following command in an Integration Console:

```
mqsi reportproperties TESTNODE_iibuser
-e default
-o HTTPConnector
-r
```

3. If CORS is enabled, you will see the following lines (amongst others):

```
corsEnabled='true'
corsAllowOrigins='*'
corsAllowCredentials='false'
corsExposeHeaders='Content-Type'
corsMaxAge='-1'
corsAllowMethods='GET, HEAD, POST, PUT, PATCH, DELETE, OPTIONS'
corsAllowHeaders='Accept, Accept-Language, Content-Language, Content-Type'
```

4. If CORS has not been enabled, run the following commands:

```
mqsic changeproperties TESTNODE_iibuser
-e default
-o HTTPConnector
-n corsEnabled -v true

mqsistop TESTNODE_iibuser

mqsistart TESTNODE_iibuser
```

2.5 Configure Integration Bus node to work with DB2

To run this lab, you should recreate any instances of the HRDB database. Follow the instructions below. You may need to alter the database schemas and authorities in the DDL to reflect your own environment.

2.5.1 Create database and tables

The HRDB database contains three tables: EMPLOYEE, EMPLOYEE_SUPPLEMENTARY and DEPARTMENT. These tables have already been created on the supplied workshop VMWare image. If you wish to create your own instance of this database (or to recreate on the workshop VM), do the following tasks:

1. Login with a user that has authority to create a new database and tables (on the workshop VM, use the user **iibadmin** (password=passw0rd).
2. Open an IIB Command Console (from the Start menu), and navigate to

```
c:\student10\Create_HR_database
```

3. Run the commands

```
1_Create_HRDB_database
```

```
2_Create_HRDB_Tables
```

4. Logout user iibadmin (or your own user).

Appropriate database permissions are included in the scripts to GRANT access to the user **iibuser**. You may need to adjust these to match your own user definitions.

2.5.2 Create JDBC and security configurable services

To run this lab, the Integration Bus node must be enabled to allow a JDBC connection to the HRDB database.

1. Login with your standard IIB developer login (**iibuser** on the workshop VM).
2. Open an IIB Integration Console (from the Start menu), and navigate to

```
c:\student10\Create_HR_database
```

3. Run the command

```
3_Create_JDBC_for_HRDB
```

Accept the defaults presented in the script. This will create the required JDBC configurable service for the HRDB database.

4. Run the command
5. Stop and restart the node to enable the above definitions to be activated. As an example, on the workshop VM:

```
mqsistop TESTNODE_iibuser
```

```
mqsisstart TESTNODE_iibuser
```

2.6 Open the Windows Log Monitor for IIB

A useful tool for IIB development on Windows is the IIB Log Viewer. This tool continuously monitors the Windows Event Log, and all messages from the log are displayed immediately.

From the Start menu, click IIB Event Log Monitor. The Monitor will open; it is useful to have this always open in the background.

The screenshot shows a window titled "IIB Event Log Monitor". The log entries are as follows:

```
BIP3132I: < IB10NODE.server1 > The HTTP Listener has started listening on port '7800', for 'http' connections. [10/3/2014 3:17:23 PM]
BIP2154I: < IB10NODE.server1 > Execution group finished with Configuration message. [10/3/2014 3:17:24 PM]
BIP2152I: < IB10NODE.server1 > Configuration message received from broker. [10/3/2014 5:07:36 PM]
BIP2153I: < IB10NODE.server1 > About to ''Change'' an execution group. [10/3/2014 5:07:36 PM]
BIP2155I: < IB10NODE.server1 > About to ''create '' the deployed resource ''EmployeeService_JSONClient'' of type ''APPZIP''. [10/3/2014 5:07:37 PM]
BIP2155I: < IB10NODE.server1 > About to ''create '' the deployed resource ''gen.getEmployee_EmployeeService_EmpServClient_JSON1'' of type ''SUBFLOW''. [10/3/2014 5:07:37 PM]
BIP2155I: < IB10NODE.server1 > About to ''create '' the deployed resource ''EmpServClient_JSON1'' of type ''MSGFLOW''. [10/3/2014 5:07:37 PM]
BIP2154I: < IB10NODE.server1 > Execution group finished with Configuration message. [10/3/2014 5:07:43 PM]
BIP3132I: < IB10NODE.HTTPListener > The HTTP Listener has started listening on port '7080' for 'http' connections. [10/3/2014 5:07:47 PM]
BIP2152I: < IB10NODE.server1 > Configuration message received from broker. [10/3/2014 5:50:41 PM]
BIP2153I: < IB10NODE.server1 > About to ''Change'' an execution group. [10/3/2014 5:50:41 PM]
BIP2155I: < IB10NODE.server1 > About to ''delete '' the deployed resource ''EmpS
```

This tool is not shipped as part of the IIB product; please contact us directly if you would like a copy.

3. Explore and start the Kafka servers

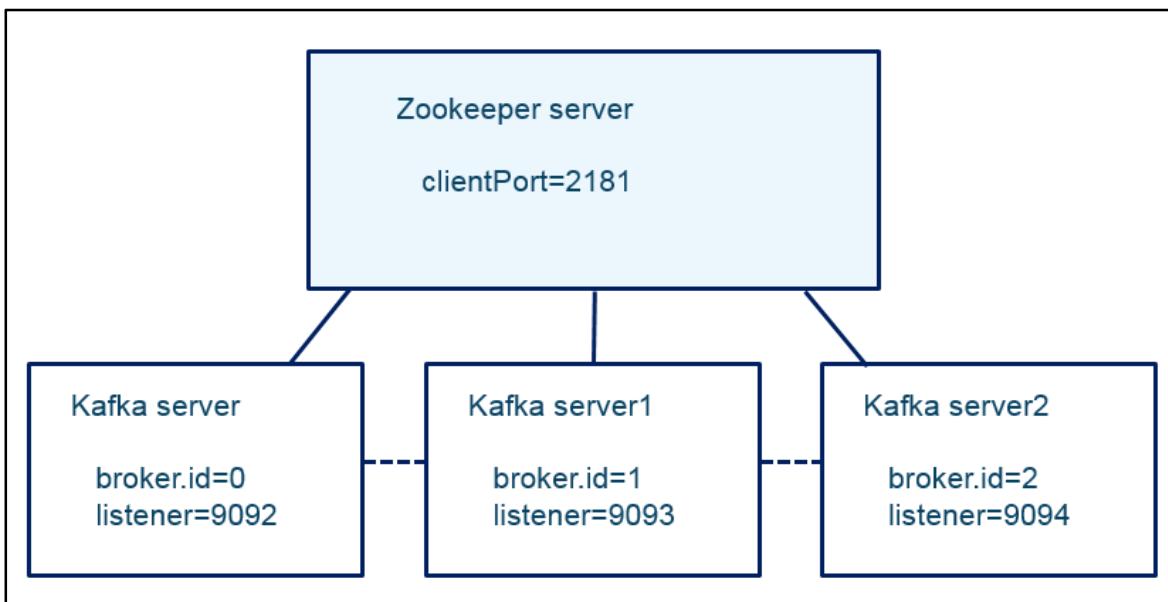
The supplied VM system has a local installation of the Apache Kafka system. The version of Kafka is 0.10.1.0. If you are running the lab scenarios on your own system, you will need to provision your own installation of Kafka. Alternatively, the sections of the lab document that reference the IBM Bluemix MessageHub Kafka implementation will work as described.

3.1 Kafka configuration for IIB workshop

On the workshop VM, Kafka is installed in **c:\tools\kafka_2.11-0.10.1.0**. In the **\bin\windows** folder, there are a number of “.bat” files that control various aspects of the Kafka system. For ease of use, some of these have been copied into the folder **c:\student10\kafka\commands**.

On the supplied workshop VM, Kafka has been configured to use a single Zookeeper server and three Kafka servers. This enables a topic Replication Factor of three. If you are doing this lab on your own system, follow the instructions below to reproduce the same configuration. You will need three Kafka servers, and a topic which has a replication factor of three.

The Kafka servers are shown schematically below. Note that all the servers are defined locally, so all have a unique listener port.



3.1.1 Kafka installation notes

The command files in **c:\student10\kafka\commands** have been tailored specifically for this lab and the workshop VMWare image. These make the various configuration items easier to achieve in this limited test environment.

If you are running this on your own system, make appropriate changes to these command files. Note that each command file sets the java CLASSPATH – make changes as appropriate for your system.

The Kafka logs have been placed in **c:\kafka**. This folder is referenced in the various “properties” files, described below.

Set the **KAFKA_HOME** variable to the install folder of kafka. Append the **KAFKA_HOME** variable to the **PATH** system environment variable.

3.2 Explore the Kafka Configuration

1. In Windows Explorer, navigate to the folder **c:\student10\kafka\config**.
2. Open the file **zookeeper.properties** using notepad++

The data directory folder has been changed for use with the workshop VM system. If you are using your own installation, make appropriate changes here.

```
#dataDir=/tmp/zookeeper
dataDir=c://kafka/zookeeper
# the port at which the clients will connect
clientPort=2181
# disable the per-ip limit on the number of connections since this is a non-production config
maxClientCnxns=0
```

Close the file when complete.

3. Open the file **server.properties**.

Most properties have been left at the default values. The following properties have been set as follows:

- Delete.topic.enable=true (allows topics to be removed at server restart)
- Broker.id=0
- Listeners=PLAINTEXT://:9092
- Log.dirs=c:/kafka/kafka-logs

If appropriate, make similar changes for your own installation, and save the file.

```
# Topic deletion properties
delete.topic.enable=true

#####
# Server Basics #####
# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0

#####
# Socket Server Settings #####
listeners=PLAINTEXT://:9092

#####
# Log Basics #####
# A comma separated list of directories under which to store log files
log.dirs=c:/kafka/kafka-logs
```

4. Make similar changes to server-1.properties and server-2.properties as necessary, as follows:

server-1.properties

- Delete.topic.enable=true
- Broker.id=1
- Listeners=PLAINTEXT://:9093
- Log.dirs=c:/kafka/kafka-logs-1

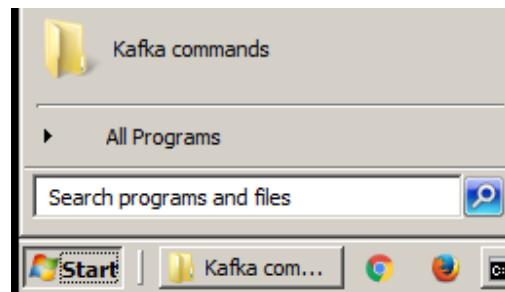
server-2.properties

- Delete.topic.enable=true
- Broker.id=2
- Listeners=PLAINTEXT://:9094
- Log.dirs=c:/kafka/kafka-logs-2

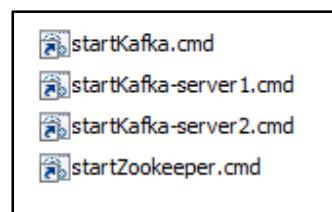
3.3 Start the Kafka servers

On the workshop VM system, Windows shortcuts have been provided for the Kafka commands that are required to start the various servers. These commands can be run as “ibuser”.

1. From the Windows Start menu (or from the desktop), open the folder “Kafka commands”



The following shortcuts will be available:



2. Open (run) **startZookeeper.cmd**. A Windows DOS command window will open and the zookeeper server will be started. A significant amount of log output will be produced.

When started this way, the "startZookeeper.cmd" name will be shown in the title line of the DOS window.

```
startZookeeper.cmd
[2016-12-14 13:01:56,977] INFO Server environment:java.io.tmpdir=C:\Users\iibuse
\<org.apache.zookeeper.server.ZooKeeperServer>
[2016-12-14 13:01:56,977] INFO Server environment:java.compiler=j9jit26 <org.apa
.ZooKeeperServer>
[2016-12-14 13:01:56,977] INFO Server environment:os.name=Windows 7 <org.apache.
KeeperServer>
[2016-12-14 13:01:56,977] INFO Server environment:os.arch=amd64 <org.apache.zook
erServer>
```

3. Open (run) **startKafka.cmd**.

As above, the server will produce some log output.

```
startKafka.cmd
[2016-12-14 13:03:27,633] INFO [ExpirationReaper-0], Starting <kafka.server.Del
ry$ExpiredOperationReaper>
[2016-12-14 13:03:27,633] INFO New leader is 0 <kafka.server.ZookeeperLeaderElec
ener>
[2016-12-14 13:03:27,633] INFO [ExpirationReaper-0], Starting <kafka.server.Del
ry$ExpiredOperationReaper>
[2016-12-14 13:03:27,664] INFO [GroupCoordinator 0]: Starting up. <kafka.coordin
er>
```

4. Repeat with **startKafka-server1.cmd** and **startKafka-server2.cmd**.

5. At this point, all Kafka servers are running, so now create a new topic.

Open a new DOS window, and change directory to **c:\student10\kafka\commands**

6. Run the **createTopic.cmd** file.

Provide the following values:

- Topic: employee
- Replication factor: 3
- Partitions 2

```
C:\student10\Kafka\commands>createTopic.cmd
C:\student10\Kafka\commands>echo off
Enter topic to create <default is employee>: employee
Enter Replication Factor for employee <default is 1>: 3
Enter number of partitions to create <default is 1>: 2
Created topic "employee".
```

7. Run the command file **listTopics.cmd**.

The command will return “employee”.

```
C:\student10\Kafka\commands>cmd /c "kafka-topics.bat --list --zookeeper localhost:2181"
employee
```

8. Run the command file **describeTopic.cmd**.

Provide “**employee**” as the topic name. The command will return information about the replication factor and partitions of the “employee” topic. If you have followed the instructions above, you will see output similar to that below.

```
C:\student10\Kafka\commands>describeTopic.cmd
C:\student10\Kafka\commands>echo off
Enter topic to describe <default is employee>: employee
Topic:employee PartitionCount:2      ReplicationFactor:3      Configs:
      Topic: employee Partition: 0    Leader: 2      Replicas: 2,0,1  Isr: 2,0,1
      Topic: employee Partition: 1    Leader: 0      Replicas: 0,1,2  Isr: 0,1,2
```

9. Run the command **consumeMessages.cmd**.

Specify the “employee” topic, and connect to the Kafka server with port 9092.

```
C:\student10\Kafka\commands>consumeMessages.cmd
C:\student10\Kafka\commands>echo off
Enter topic that you want to consume from <default is employee>: employee
Enter port that you want to connect to <server=9092, server1=9093, server2=9094, default is 9092>: 9
092
```

10. Open a further DOS windows, and navigate to c:\student10\kafka\commands.

Run the command **produceMessage.cmd**.

Specify the “employee” topic, and connect to Kafka with port 9092.

Type some text message input, as shown below. Each message is terminated with the Return key.

```
C:\student10\Kafka\commands>produceMessage.cmd
C:\student10\Kafka\commands>echo off
Enter topic you want to produce messages to <default is employee>: employee
Enter port that you want to connect to <server=9092, server1=9093, server2=9094, default is 9092>: 9
092
test message 1
test message 2
final message
```

11. Back in the consumeMessages window, observe that the text messages you just produced have been consumed by the consumeMessages client application.

```
C:\student10\Kafka\commands>consumeMessages.cmd
C:\student10\Kafka\commands>echo off
Enter topic that you want to consume from <default is employee>: em
Enter port that you want to connect to <server=9092, server1=9093,
092
test message 1
test message 2
final message
```

You have now verified that the local Kafka system is configured correctly, and can be used by the IIB applications.

4. Import and extend the HR_Service REST API

In this section you will extend the provided REST API, HR_Service. The provided version of HR_Service has already implemented the two createEmployee* operations. Both these operations use the createEmployeeMain.subflow. This subflow is the component that will be extended in this section.

1. If you already have a workspace open, click File, Switch Workspace. Give the new workspace the name

```
c:\users\iibuser\IBM\IIBT10\workspace_kafka
```

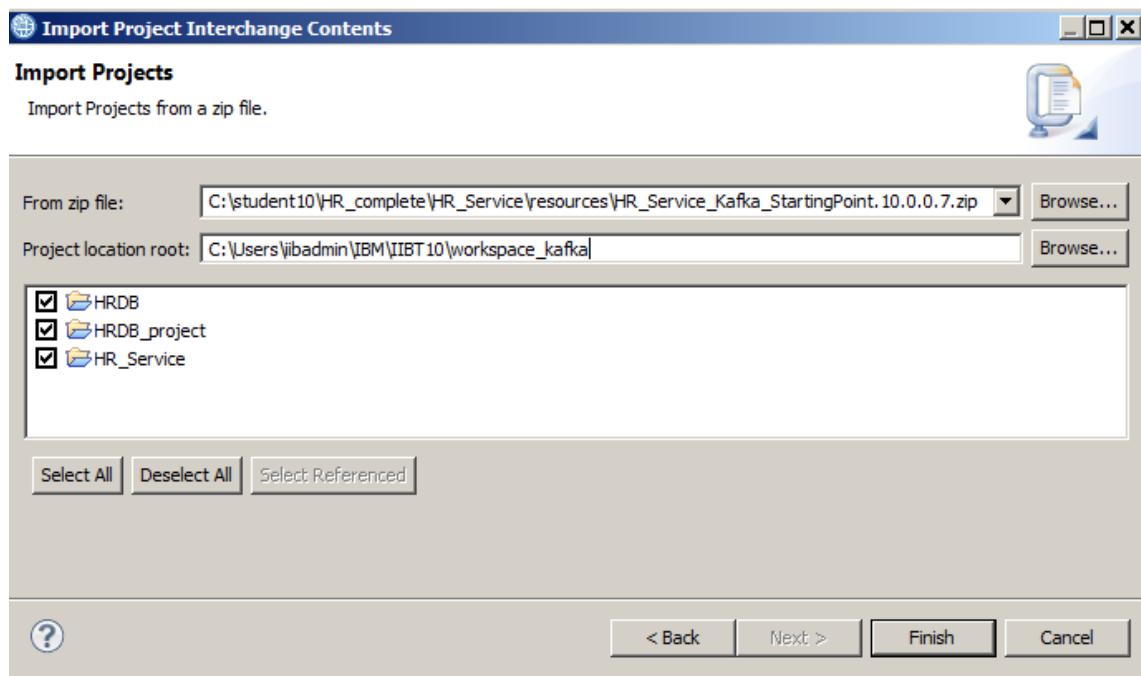
2. In the new workspace, import the Project Interchange file:

```
C:\student10\HR_complete\HR_Service\resources\HR_Service_Kafka_StartPoint.10.0.0.7.zip
```

Select all three projects from this PI file, and click Finish.

HR_Service uses the EMPLOYEE, EMPLOYEE_SUPPLEMENTARY and DEPARTMENT tables from the HRDB database. This requires the HRDB Database Definition project, which represents the tables schemas. This is used by the Mapping nodes that access these tables. The HRDB Shared Library and HRDB_project items contain the database definitions for the DB2 database HRDB.

Note – because the EMPLOYEE_SUPPLEMENTARY table has been added in the 10.0.0.7 version of this lab series, the HRDB.dbm file has been updated in this version of the HRDB shared library. However, it is backward-compatible with earlier versions, and can be used with earlier lab scenarios from this series.

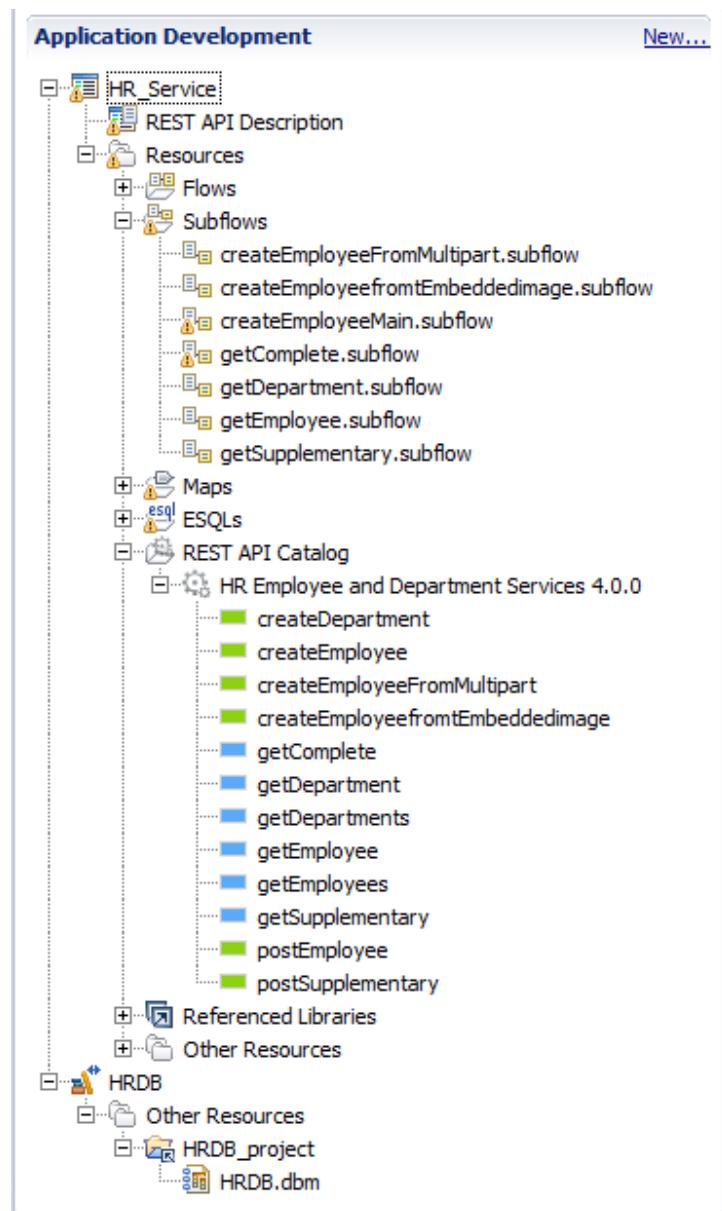


3. When imported, you will see the HR_Service REST API project, and the HRDB shared library. The shared library has a library reference from the HR_Service REST API.

You will see several subflows are present in HR_Service. This indicates that several operations have already been implemented in this REST API. In addition, there are other subflows that are for more general use.

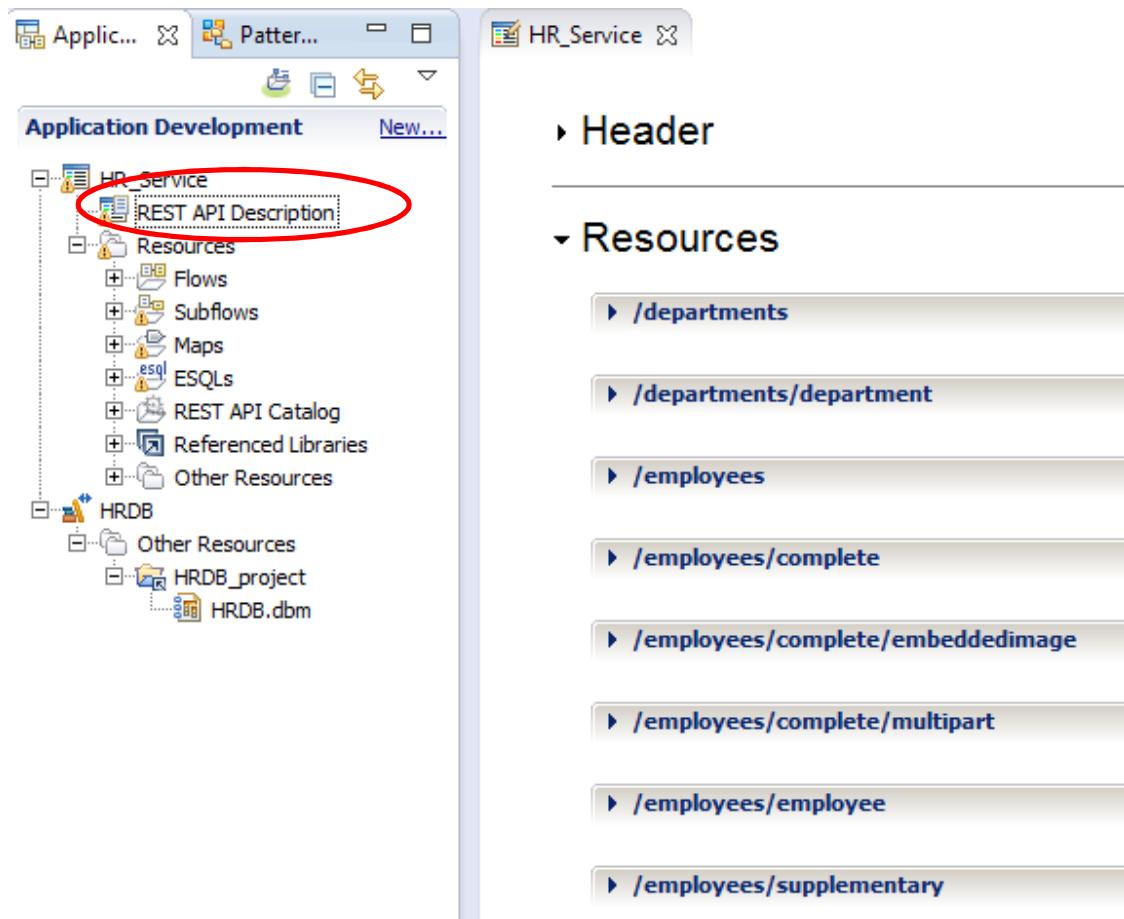
Expanding the REST API Catalog will show the entire list of operations that are defined in this REST API. As mentioned above, not all of these operations have been implemented.

Note that the “HR Employee and Department Services.json” document has been updated to Version 4.0.0. This is because new Model Definitions have been provided for the new database table, and all “Get” operations have been changed to use the “query parameter” form of a REST request with a “get” method.



4. Open (double-click) the REST API Description. This will show the Header and Resources definitions in the REST API.

The “get” operations are not described here. If you need information on how to construct these operations, please refer to various lab documents earlier in this series, for example 10006_16L01/02.



5. Further down the HR_Service editor, you will see the supplied Model Definitions.

Expand the supplied models. These will be used later in the lab.

▼ Model Definitions

Name	Array	Type	Allow null
(+) <Enter a unique name to create a new model>			
[+] {...} EMPLOYEE	<input type="checkbox"/>	object	
[+] {...} DEPARTMENT	<input type="checkbox"/>	object	
[+] {...} DBRESP	<input checked="" type="checkbox"/>	object	
[...] EmployeeResponse	<input type="checkbox"/>	object	
DBResp	<input type="checkbox"/>	DBRESP	
[] Employee	<input checked="" type="checkbox"/>	EMPLOYEE	
[...] DepartmentResponse	<input type="checkbox"/>	object	
[...] CompleteResponse	<input type="checkbox"/>	object	
DBResp_employee	<input type="checkbox"/>	DBRESP	
Employee	<input type="checkbox"/>	EMPLOYEE	
DBResp_department	<input type="checkbox"/>	DBRESP	
Department	<input type="checkbox"/>	DEPARTMENT	
DBResp_employee_supplementary	<input type="checkbox"/>	DBRESP	
Employee_supplementary	<input type="checkbox"/>	EMPLOYEE_SUPPLEMENTARY	
[...] EMPLOYEE_SUPPLEMENTARY	<input type="checkbox"/>	object	
EMPNO_SUPP	<input type="checkbox"/>	string	
EMAIL	<input type="checkbox"/>	string	
MOBILEPHONE	<input type="checkbox"/>	string	
TWITTERID	<input type="checkbox"/>	string	
BOXID	<input type="checkbox"/>	string	
IMAGE	<input type="checkbox"/>	string	
[...] EmployeeSupplementaryResponse	<input type="checkbox"/>	object	
DBResp	<input type="checkbox"/>	DBRESP	
Employee_supplementary	<input type="checkbox"/>	EMPLOYEE_SUPPLEMENTARY	
[...] EmployeeAddUpdateCompleteRequest	<input type="checkbox"/>	object	
Employee	<input type="checkbox"/>	EMPLOYEE	
EmployeeSupplementary	<input type="checkbox"/>	EMPLOYEE_SUPPLEMENTARY	

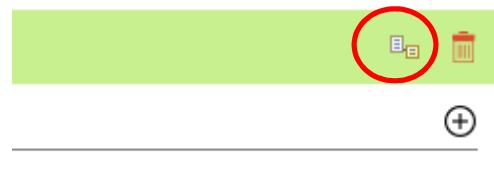
6. Expand the /employees/complete/embeddedimage resource.

You will see a POST operation has been created in this resource.

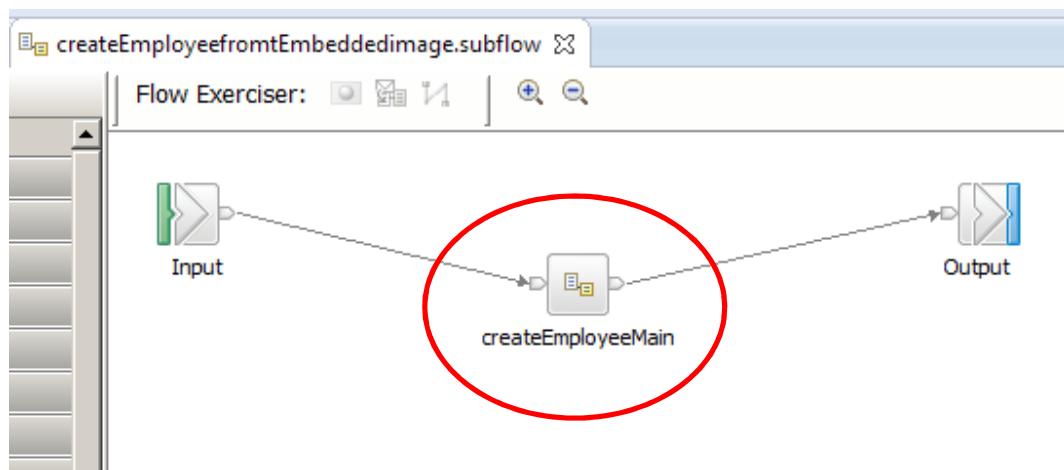
The screenshot shows the configuration of a POST operation named 'createEmployeefromEmbeddedimage' for the resource '/employees/complete/embeddedimage'. The operation details are as follows:

Name	Parameter type	Data type	Format	Required	Description
Request body					
The request body for the operation				JupdateCompleteRequest	Schema type
Response status Description					
200	The operation was successful.				

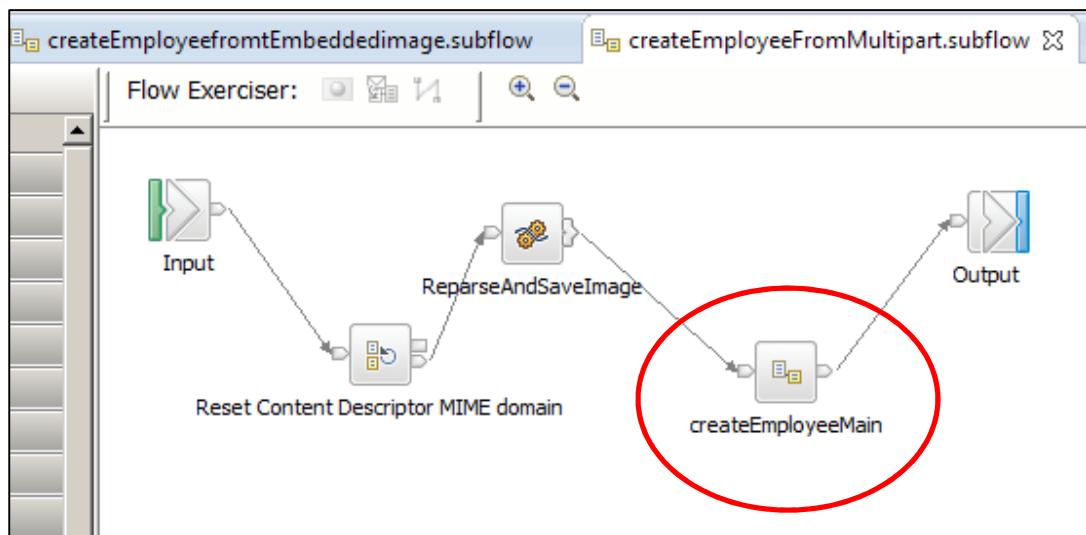
7. Use the slide bar to move to the right, and open the subflow that implements this operation.



8. The subflow invokes a further subflow, createEmployeeMain, where the main logic for this operation is provided.



9. Similarly, in the **/employees/complete/multipart** resource, the **createEmployeeFromMultipart** operation also invokes the same subflow.

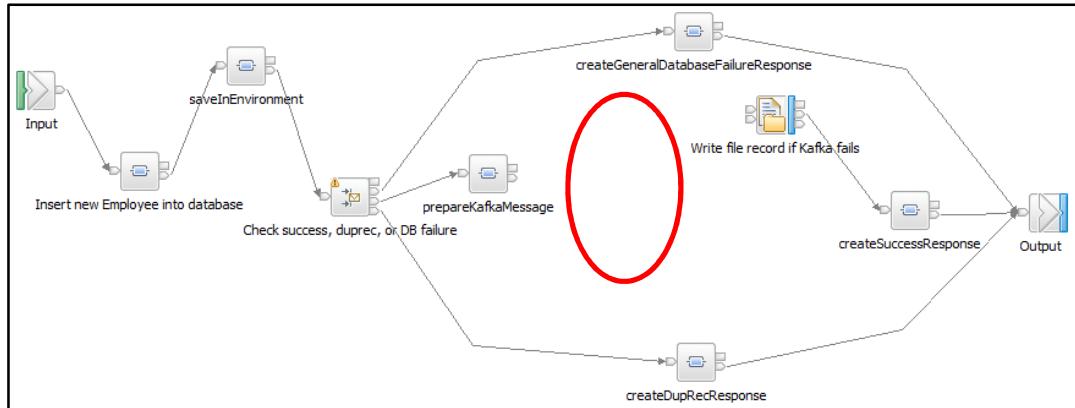


4.1 Investigate the createEmployeeMain subflow

- From either of the two operation subflows mentioned above, open (double-click) the createEmployeeMain subflow.

This subflow is complete, with the exception of a space for a KafkaProducer node.

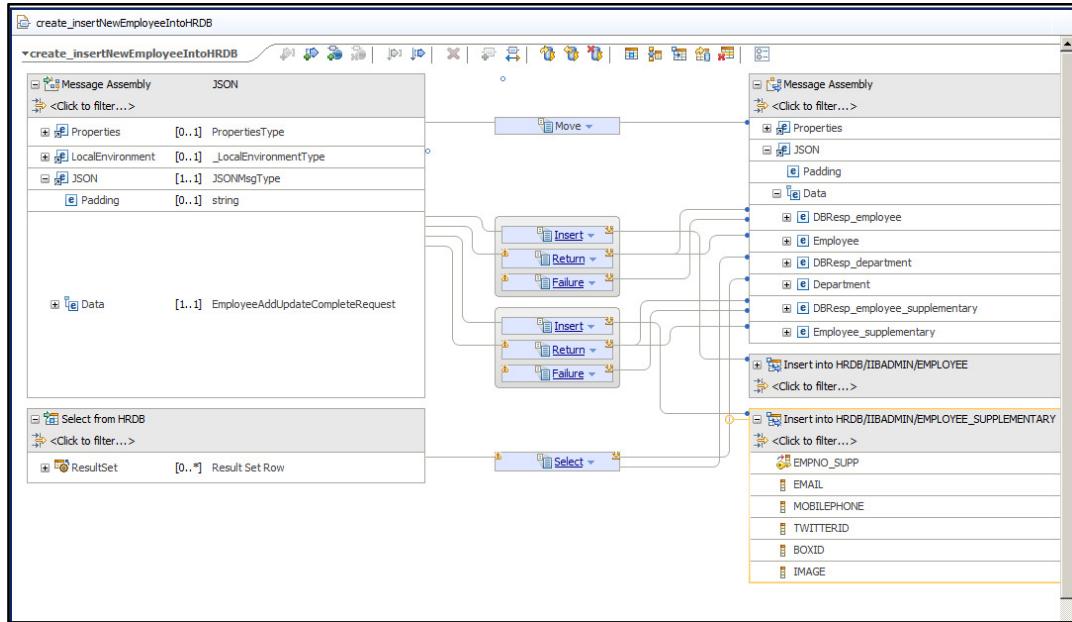
However, there are some additional features of some other nodes which will be explored, although no further development work is required for these.



2. Open the “Insert new Employee into database” mapping node. This map accesses three database tables, all in the HRDB database, as follows:

- Select from HRDB(DEPARTMENT)
- Insert into HRDB(EMPLOYEE)
- Insert into HRDB(EMPLOYEE_SUPPLEMENTARY)

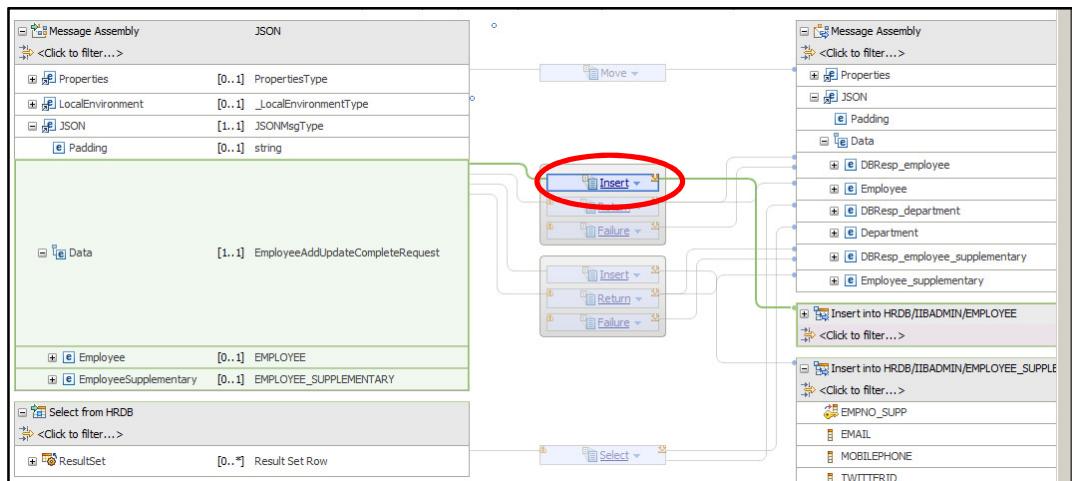
If all database operations are successful, all data is placed in the output message assembly, with the exception of the binary image element of the input message (see step 6 below).



3. Highlight the first Insert transform. The input Data element and the output “Insert into HRDB/IIBADMIN/EMPLOYEE” elements are highlighted.

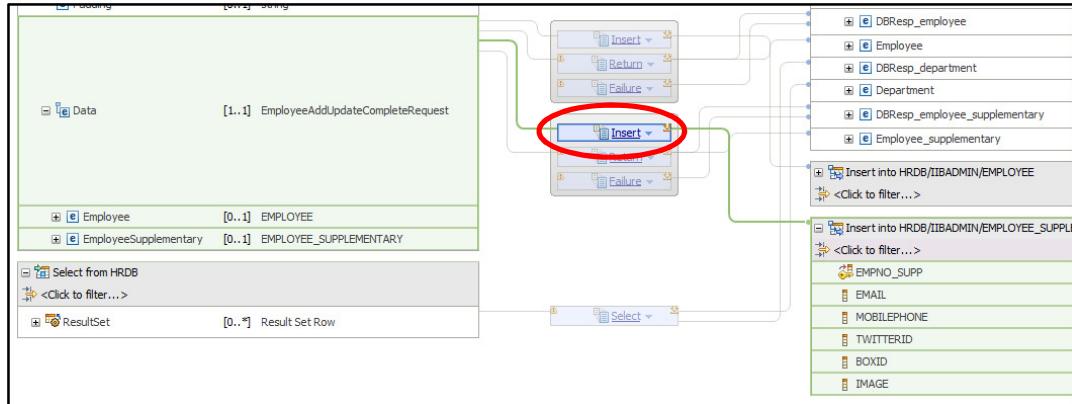
If you want to see the precise elements mappings, click the “Insert” text of the transform.

Return to the highest level of the map when finished.

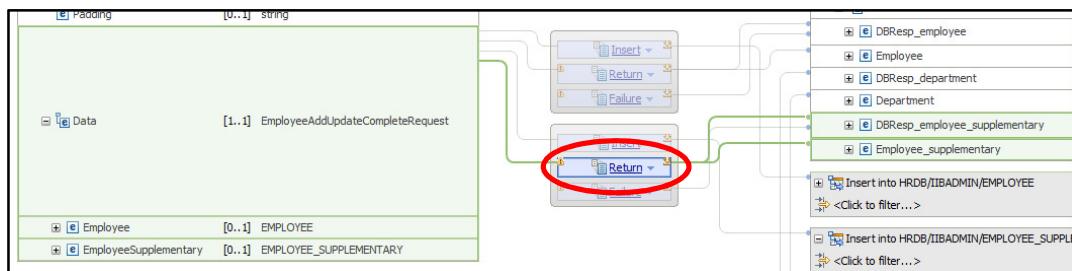


4. Similarly, highlight (click) the second Insert transform (not the “Insert” text). The same input will be highlighted, and the output assembly for the EMPLOYEE_SUPPLEMENTARY table will be highlighted.

Again, if required, click the word “Insert” to see the precise mappings.

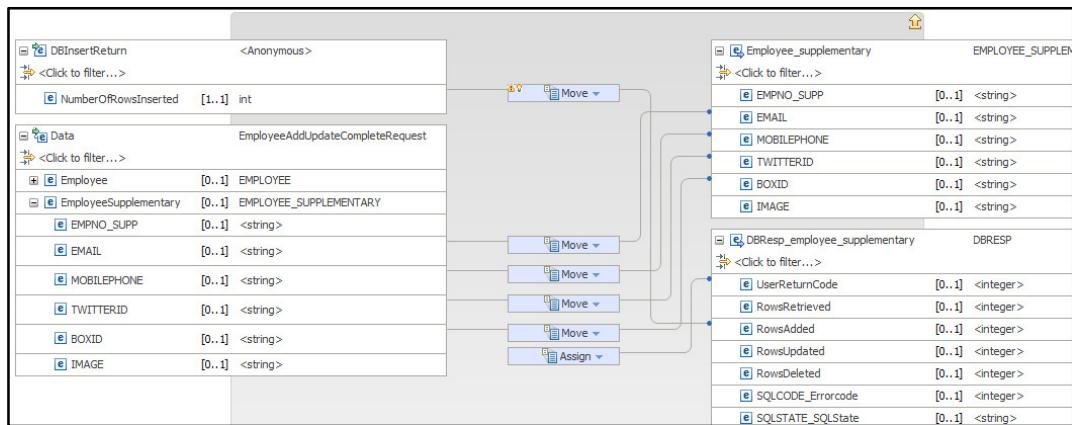


5. Highlight and click the Return transform shown. Note the input and output connections that have been made for this transform.



6. Finally in this map, click the word “Return” to see the precise element mappings.

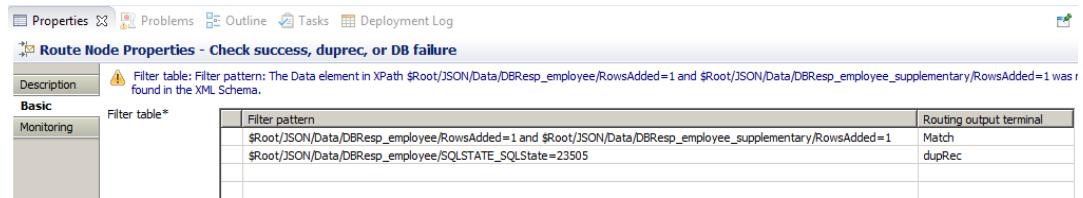
Note that the IMAGE element is not mapped. This is because this element will probably be quite large, and we don't want to send this back to the originating client.



Close the map.

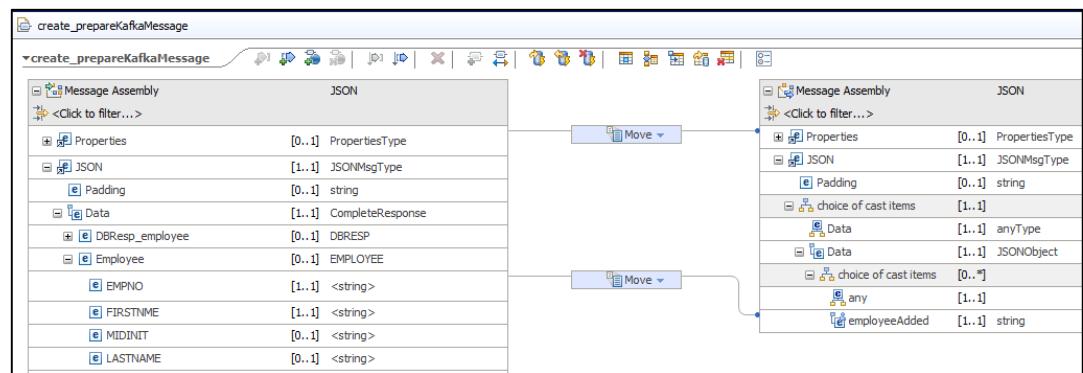
7. Highlight the “Check success, duprec, or DB failure“ Route node.

In the properties of this node, the filter pattern has been set to check for successful inserts to both the EMPLOYEE and EMPLOYEE_SUPPLEMENTARY tables. If RowsAdded=1 for both of these tables, then the message will be sent to the Match terminal, and to the KafkaProducer node (to be added). All other responses represent failures of various kinds.



8. Open the prepareKafkaMessage map.

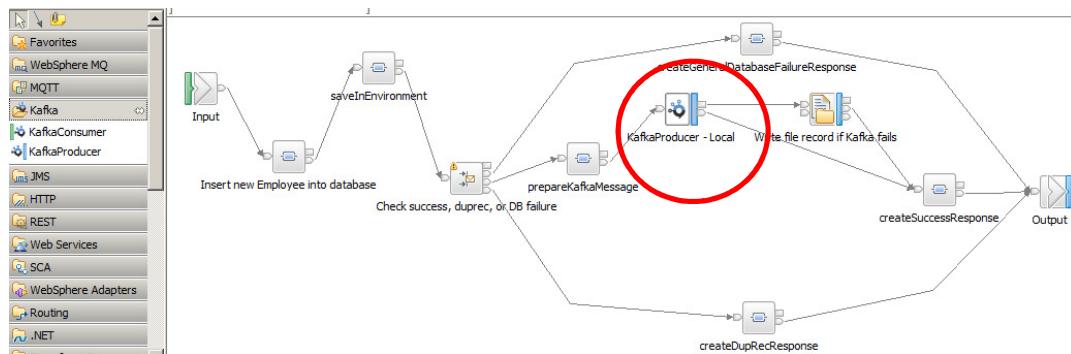
The output message assembly has a single JSONObject element, employeeAdded. This element is set to the value of the employee number (EMPNO), before being published to the Kafka server in the subsequent node. Consuming applications will receive this small message, and be able to retrieve the full information from the EMPLOYEE and EMPLOYEE_SUPPLEMENTARY tables.



Close the map.

9. From the Kafka drawer, drop a KafkaProducer node onto the subflow, name it "KafkaProducer - Local", and connect as shown.

Connect the Failure terminal to the In terminal of the FileOutput node.



10. In the properties of the KafkaProducer node, set the following properties:

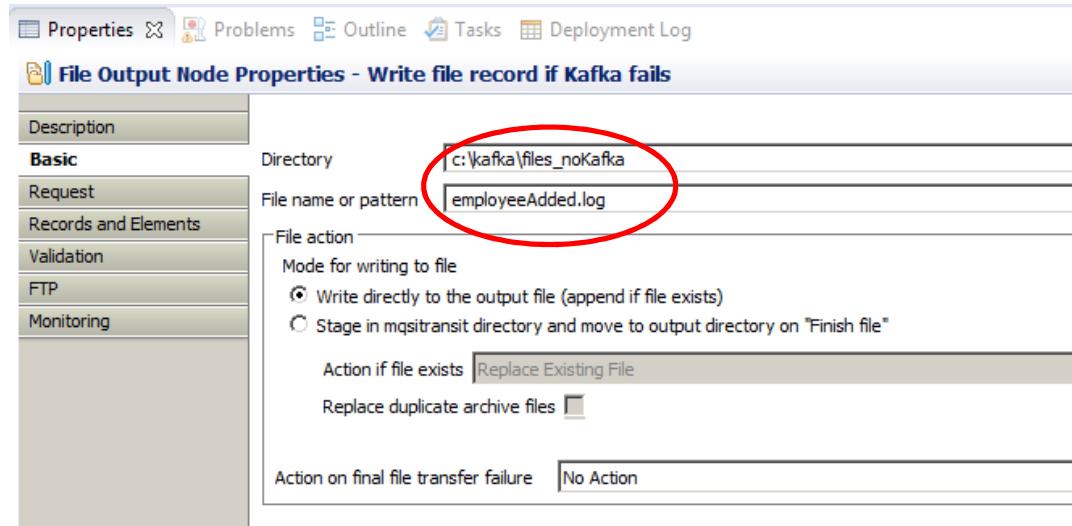
- Topic: employee
- Bootstrap server: localhost:9092
- Acks: 1

Properties		Problems	Outline	Tasks	Deployment Log
KafkaProducer Node Properties - KafkaProducer - Local					
Description					
Basic	Topic name*	employee			
Security	Bootstrap servers*	localhost:9092			
Validation	e.g. bootstrap.server.com:9092 (multiple servers can be specified and delimited using a ',')				
Monitoring	Client ID				
	Add IIB suffix to client ID	<input checked="" type="checkbox"/>			
	Acks*	1			
	Timeout (sec)*	60			

11. Review the properties of the FileOutput node.

The properties have been specified to work on a Windows system, and to write an output file to the folder **c:\kafka\files_noKafka**, if the KafkaProducer node should fail.

If you are running on a non-Windows system, or you want to change the output folder destination or file name, make appropriate changes now to these properties.



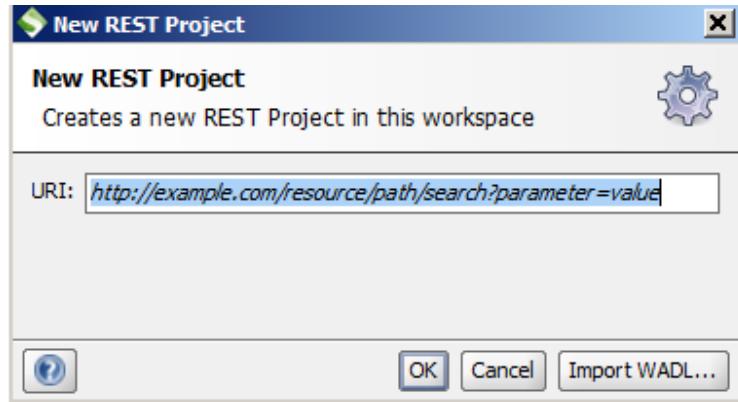
12. Save the subflow.

Deploy the **HRDB** Shared Library and then the **HR_Service** REST API to TESTNODE_iibuser/default in the usual way.

5. Test the updated HR_Service REST API

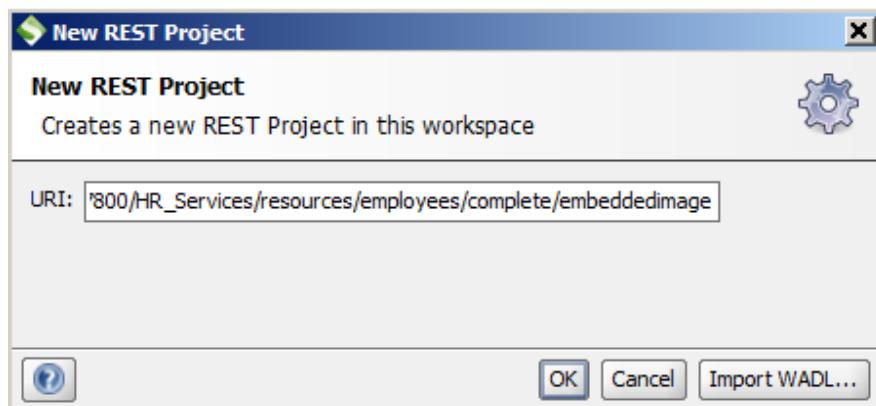
5.1 Using SOAPUI

1. Open SOAPUI and create a new REST project.



2. In the URI field, specify the following URI, and click OK.

`http://localhost:7800/HR_Services/resources/employees/complete/embeddedimage`



3. Change the method to a POST.

In the input data area, paste the entire contents of the file

```
c:\student10\HR_complete\HR_Service\data\  
EmbeddedImageWholeMessage.raw.json
```

Note that the last data line (highlighted in yellow below) is a binary image of the new employee, encoded as a Base64 text encoded string.

RE
ST Request 1

Method Endpoint

POST http://localhost:7800

Request

Name	Value	Style	Level
------	-------	-------	-------

Raw

Required: Sets if parameter is required

Type:

Options:

Media Type application/json

{
 "Employee": {
 "EMPNO": "000015",
 "FIRSTNAME": "Albert",
 "MIDINIT": "J",
 "LASTNAME": "Einstein",
 "WORKDEPT": "A00",
 "PHONE": "5012",
 "HIREDATE": "2016-12-12",
 "JOB": "MGR",
 "EDLEVEL": 9,
 "SEX": "M",
 "BIRTHDATE": "1879-03-14",
 "SALARY": 20000,
 "BONUS": 5000,
 "COMM": 1024
 },
 "EmployeeSupplementary": {
 "EMPNO_SUPP": "000015",
 "EMAIL": "a.einstein@zurich.com",
 "MOBILEPHONE": "7845364535",
 "TWITTERID": "@EMC2",
 "BOXID": "EMC2@box.com",
 "IMAGE":
 "/9j/4AAQSkZJRgABAgAAZABkAAD/7ABXRHVja3kAAQAEAAAAPAADE"
 }
}

4. Change the EMPNO to an employee number that does not already exist in the EMPLOYEE table (000028 in this example).

Click the green arrow to send the request.

The screenshot shows the 'Request 1' interface in the IBM Integration Bus. The 'Method' is set to 'POST' and the 'Endpoint' is 'http://localhost:7800'. The 'Raw' tab is selected, showing a JSON payload:

```
{  
    "Employee": {  
        "EMPNO": "000028",  
        "FIRSTNAME": "Albert",  
        "MIDINIT": "J",  
        "LASTNAME": "Einstein",  
        "WORKDEPT": "A00",  
        "PHONE": "5012",  
        "HIREDATE": "2016-12-12",  
        "JOB": "MGR",  
        "EDLEVEL": 9,  
        "SEX": "M",  
        "BIRTHDATE": "1879-03-14",  
        "SALARY": 20000,  
        "BONUS": 5000,  
        "COMM": 1024  
    },  
}
```

5. If successful, the response message will be shown.

The screenshot shows the IBM Integration Bus interface with a response message displayed. The message is an XML document with the following structure:

```

<Response xmlns="http://localhost/HR_Services/resources/employees/complete/embeddedimage">
  <DBResp_department>
    <RowsRetrieved>1</RowsRetrieved>
    <UserReturnCode>0</UserReturnCode>
  </DBResp_department>
  <DBResp_employee>
    <RowsAdded>1</RowsAdded>
    <UserReturnCode>0</UserReturnCode>
  </DBResp_employee>
  <DBResp_employee_supplementary>
    <RowsAdded>1</RowsAdded>
    <UserReturnCode>0</UserReturnCode>
  </DBResp_employee_supplementary>
  <Department>
    <ADMRDEPT>A00</ADMRDEPT>
    <DEPTNAME>SPIFFY COMPUTER SERVICE DIV.</DEPTNAME>
    <DEPTNO>A00</DEPTNO>
    <LOCATION null="true"/>
    <MGRNO>000010</MGRNO>
  </Department>
  <Employee>
    <BIRTHDATE>1879-03-14</BIRTHDATE>
    <BONUS>5000</BONUS>
    <COMM>1024</COMM>
    <EDLEVEL>9</EDLEVEL>
    <EMPNO>000028</EMPNO>
    <FIRSTNME>Albert</FIRSTNME>
    <HIREDATE>2016-12-12</HIREDATE>
  </Employee>

```

The interface includes tabs for XML, JSON, HTML, and Raw, with XML selected. The URL bar at the top shows the path: /HR_Services/resources/employees/complete/embeddedimage.

6. Switch back to the DOS command window which was consuming messages from the Kafka employee topic.

Note that a message has been received with the employee number of the newly added employee (000028 in this example).

```
C:\student10\Kafka\commands>echo off
Enter topic that you want to consume from <default is employee>: employee
Enter port that you want to connect to <server=9092, server1=9093, server2=9092>
test message 1
test message 2
final message
{"employeeAdded":"000026"}
{"employeeAdded":"000027"}
{"employeeAdded":"000028"}
```

5.2 Using Postman

Although most REST testing tools, such as SOAPUI, are capable of generating and sending a REST request with an accompanying JSON payload, not all are capable of sending a MIME request in multipart format. The HR_Service REST API has two operations to create a new employee. The second of these expects such a message, so this section uses the Chrome Postman tool to send this request. Postman is described in more detail in the lab guide 10006_16L15 document.

1. Open the Postman tool from the Start menu and import the Postman collection “Complete” (depending on your VM image this may already be there – if so, replace it). This is available in the export file

```
c:\student10\HR_complete\HR_Service\postman\  
Complete.postman_collection.json
```

2. Expand Complete, and select the “Add (Complete) – using multipart image” test.

On the details pane, select the Body tab. The data format will be set to “form-data”.

The screenshot shows the Postman application interface. The left sidebar lists collections: 'Complete' (selected), 'EmbeddedImage', and 'Multipart'. The main area shows a POST request for 'Add (Complete) - using multipart image' to 'http://localhost:7800/HR_Services/re'. The 'Body' tab is selected in the configuration pane, which displays two form-data fields: 'employeeData' (Text) and 'employeeImage' (File).

3. For the **employeeImage** part of the message, set the attached file to

c:\student10\HR_complete\HR_service\data\Einstein.jpg

Set the EMPNO element to an employee value that is not yet in the database. The example below shows "000029".

The screenshot shows the Postman interface with a POST request to `http://localhost:7800/HR_Services/resources/employees/complete/multi`. The 'Body' tab is selected, showing 'form-data' selected. Two fields are present: 'employeeData' with value `{ "Employee": { "EMPNO": "000029" }}` and 'employeeImage' with value `Einstein.jpg` chosen from a file browser. The 'key' field is also visible. The 'Send' button is highlighted.

4. Click Send.

The new employee will be added successfully.

The new employee will be published to Kafka, and the consuming application will receive the new notification, as before.

```
C:\student10\Kafka\commands>echo off
Enter topic that you want to consume from <default is employee>: employee
Enter port that you want to connect to <server=9092, server1=9093, server2=9094, 092
test message 1
test message 2
final message
{"employeeAdded":"000026"}
{"employeeAdded":"000027"}
 {"employeeAdded":"000028"}
 {"employeeAdded":"000029"}
```

6. Using the Kafka nodes with IBM MessageHub

The IIB Kafka Producer and Consumer nodes can also be used with the MessageHub service provided on IBM Bluemix. MessageHub is IBMs implementation of Apache Kafka. More information can be found here: <https://developer.ibm.com/messaging/message-hub/>.

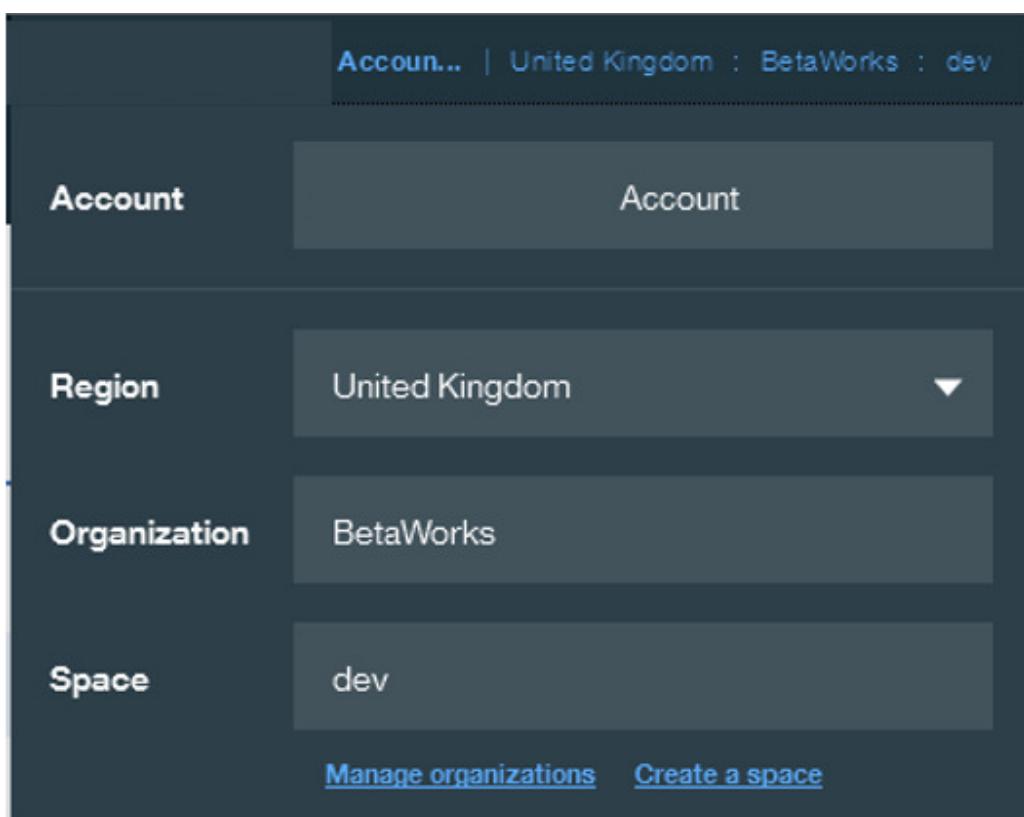
6.1 Explore and configure MessageHub

1. To use the Bluemix MessageHub service, you will need to login with an IBM ID. If you do not have an IBM ID, create one now, and then return to the next step.

2. Login to the MessageHub service on IBM Bluemix:

`https://console.ng.bluemix.net/catalog/services/message-hub`

When you have logged in, make sure that the Bluemix Region is set to the value that you want to use for your own testing. In the example below, the region for the BetaWorks organization is “United Kingdom”.



3. In the Bluemix Catalog, type “Message Hub” into the search field.

Click the Message Hub icon.

All Categories (1) >

message hub

Catalog Support Account

Infrastructure

- Compute
- Storage
- Network
- Security

Services

Application Services

Deliver new web and mobile apps.

Message Hub
IBM Message Hub
is a scalable,
IBM

4. Click Create to create a new Message Hub service.

Message Hub

IBM Message Hub is a scalable, high-throughput message bus. Wire micro-services together using open protocols. Connect stream data to analytics to realize powerful insights. Feed event data to multiple applications to react in real time. Bridge to your on-premise messaging infrastructure to create a hybrid cloud messaging solution.

Service name: Message Hub-a3

Credential name: Credentials-1

Features

- Fast, scalable, fully managed messaging service, based on Apache Kafka**
Message Hub sits on Apache Kafka, an open-source, high-throughput messaging system which provides a low-latency platform for handling real-time data feeds.
- Streaming analytics integration**
Use the power of the Apache Spark and Streaming Analytics services alongside Kafka to build high-performance scalable streaming analytics solutions.
- Communication using Kafka, MQ Light or REST API clients**
Message Hub provides 3 interfaces through which messages can be produced and consumed. The native Kafka interface for Kafka clients, the MQ Light API for MQ Light clients, and a REST API.
- Connectivity with a range of services**
Message Hub will integrate with multiple services on Bluemix.

Connect to: Leave unbound

[View Docs](#)

AUTHOR IBM
PUBLISHED 06/23/2016
TYPE Service
LOCATION United Kingdom

Need Help? [Contact Bluemix Sales](#)

Estimate Monthly Cost [Cost Calculator](#)

Create

5. The new Message Hub service will be created. The views may look slightly different but should be similar to what is shown below.

The screenshot shows the 'Manage' tab selected in the top navigation bar. Below it, the 'Topics' section is active. A search bar labeled 'Filter Topics...' with a magnifying glass icon is present. Below the search bar, a message states 'Partitions: 0 used / 0 maximum'. There are four icons: a square, a circular arrow, a plus sign, and a trash can. A table header row is shown with columns: 'Topic Names', 'Partitions', and 'Retention (hours)'. A message at the bottom of the table area says 'No Topics Found'.

6. Create a new topic, by clicking the “Create topic” icon.

This screenshot is similar to the previous one, showing the 'Topics' section. However, the 'Create topic' button in the 'Topic Names' column is highlighted with a blue arrow pointing to it. The rest of the interface elements are identical to the first screenshot.

7. Name the new topic "employee" and press Return.

Topic Names	Partitions	Retention (hours)
<input type="text" value="employee"/>	<input type="text" value="1"/>	<input type="text" value="24"/>
No Topics Found		

The new topic will be created with a retention period of 24 hours.

Topic Names	Partitions	Retention (hours)
Topic 'employee' created.		
<input type="checkbox"/> employee	1	24

8. Click the "Service Credentials" tab.

Click the "View Credentials" drop-down.

Note the server URL values, and the user and password values. These will be used by the IIB KafkaProducer node that you update shortly.

Message Hub-a3

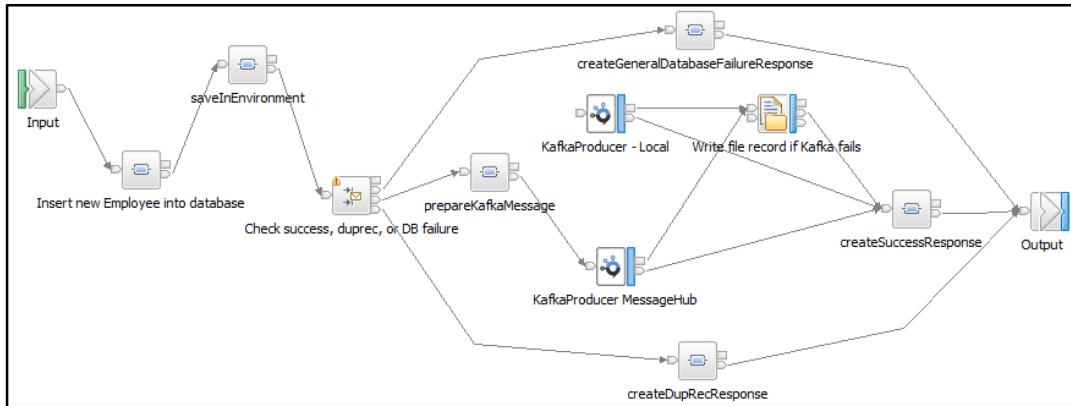
Manage **Service Credentials** Connections

Service Credentials		
Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.		
<input type="checkbox"/> KEY NAME	DATE CREATED	ACTIONS
<input type="checkbox"/> Credentials-1	Dec 14, 2016 - 05:05:39	View Credentials ▾
<pre>{ "mqlight_lookup_url": "https://mqlight-lookup-prod02.messagehub.services.eu-gb.bluemix.net/Lookup?serviceId=845afeed-a856-4726-a2c5-af90b2e7eb37", "api_key": "yt3ApcyClTbrJ7H9nb0zXz466PH0cRvDmNahhwZ3FdBgJoRl", "kafka_admin_url": "https://kafka-admin-prod02.messagehub.services.eu-gb.bluemix.net:443", "kafka_rest_url": "https://kafka-rest-prod02.messagehub.services.eu-gb.bluemix.net:443", "kafka_brokers_sasl": ["kafka01-prod02.messagehub.services.eu-gb.bluemix.net:9093", "kafka02-prod02.messagehub.services.eu-gb.bluemix.net:9093", "kafka03-prod02.messagehub.services.eu-gb.bluemix.net:9093", "kafka04-prod02.messagehub.services.eu-gb.bluemix.net:9093", "kafka05-prod02.messagehub.services.eu-gb.bluemix.net:9093"], "user": "yt3ApcyClTbrJ7H9", "password": "nb0zXz466PH0cRvDmNahhwZ3FdBgJoRl" }</pre>		

9. Return to the Integration Toolkit, and edit the createEmployeeMain subflow.

Add a new KafkaProducer node to the flow. Name it “KafkaProducer MessageHub”, and connect it as shown.

Be sure to remove the connection to the KafkaProducer Local node – or you will get logic errors at runtime.



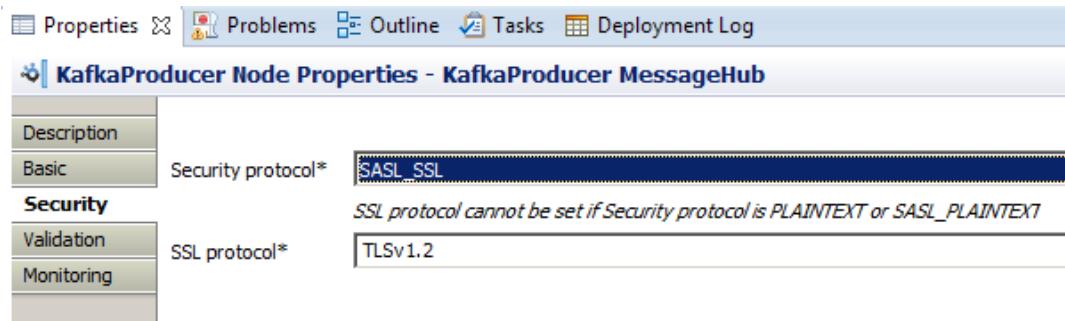
10. Select the properties of the KafkaProducer MessageHub node.

On the Basic tab:

- Topic: employee
- Bootstrap server: kafkabluemix.net:9093 (copy/paste the first url from the Service Credentials from your Bluemix MessageHub service)
- Acks: 1

Properties		Problems	Outline	Tasks	Deployment Log
KafkaProducer Node Properties - KafkaProducer MessageHub					
Description					
Basic	Topic name*	employee			
Security	Bootstrap servers*	kafka01-prod02.messagehub.services.eu-gb.bluemix.net:9093			
Validation	Client ID				
Monitoring	Add IIB suffix to client ID	<input checked="" type="checkbox"/>			
	Acks*				
	Timeout (sec)*	60			

11. On the Security tab, set the security protocol to SASL_SSL.



12. When connecting to a secure Kafka server, such as MessageHub, the required security credentials are provided by the IIB node. Security credentials are created with the "mqsisetdbparms" command.

In an IIB Command Console, run the following command, substituting values for your IIB server, username and password as appropriate (user and password are those provided by the MessageHub credentials).

```
mqsisetdbparms TESTNODE_iibuser
-n kafka::KAFKA::default
-u yt3ApcyC1TbrJ7H9
-p nb0zXz466PH0cRvDmNAhhwZ3FdBgJoRl
```

13. Stop and restart the IIB node.

14. Save the subflow, and deploy the updated HR_Service in the usual way.

6.2 Test HR_Service with MessageHub

1. Return to the SOAPUI application that was used earlier. Change the EMPNO to a value that has not been previously used.

Click the green arrow to invoke.

The screenshot shows the SOAPUI interface with a request named "Request 1". The request details are as follows:

- Method:** POST
- Endpoint:** <http://localhost:7800>

The "Raw" tab contains the following JSON payload:

```
{  
    "Employee": {  
        "EMPNO": "000031",  
        "FIRSTNAME": "Albert",  
        "MIDINIT": "J",  
        "LASTNAME": "Einstein",  
        "WORKDEPT": "A00",  
        "PHONE": "5012",  
        "HIREDATE": "2016-12-12",  
        "JOB": "MGR",  
        "EDLEVEL": 9,  
        "SEX": "M",  
        "BIRTHDATE": "1879-03-14",  
        "SALARY": 20000,  
        "BONUS": 5000,  
        "COMM": 1024  
    }  
}
```

2. Note that to connect a client directly to the MessageHub system, you can use a variety of client tools. In all cases, you will have to configure appropriate security components to enable SSL or SASL-style connectivity, which is beyond the scope of this lab. You may wish to explore further aspects of the Bluemix Message Hub user interface such as the Grafana tools for example.

Topic Names	Partitions	Retention (hours)
employee	1	24

Metrics & Logs

Grafana provides metrics, dashboards and graphs for Message Hub:

[GRAFANA](#)

END OF KAFKA LAB SCENARIO

7. LOOPBACK LAB: Introduction and Preparation

7.1 Introduction

IIB v10.0.0.6 introduced a new Loopback Request node, and this can be used to access databases such as MongoDB.

This lab provides a basic example of retrieving information from a MongoDB database. It is based on a previously built REST API, and will retrieve Employee documents from the HRDB database (EMPLOYEE container), located within a MongoDB instance.

IIB v10, fixpack 5 (10.0.0.5) has made significant changes in the Toolkit representation of REST APIs, the editor functions, and with the tools provided with the Mapping Node editor. This lab will make use of those functions to extract the required employee number from the input parameters. An approach is also provided to retrieve the same information using ESQL.

7.2 Open the Windows Log Monitor for IIB

A useful tool for IIB development on Windows is the IIB Log Viewer. This tool continuously monitors the Windows Event Log, and all messages from the log are displayed immediately.

From the Start menu, click IIB Event Log Monitor. The Monitor will open; it is useful to have this always open in the background.

The screenshot shows a Windows application window titled "IIB Event Log Monitor". The window contains a list of log entries, each consisting of a timestamp, a log ID, and a message. The log entries are color-coded: green for informational messages and red for error messages. The messages relate to the startup and configuration of an IIB node named "IB10NODE.server1". Key messages include:

- BIP3132I: < IB10NODE.server1 > The HTTP Listener has started listening on port '7800' for 'http' connections. [10/3/2014 3:17:23 PM]
- BIP2154I: < IB10NODE.server1 > Execution group finished with Configuration message. [10/3/2014 3:17:24 PM]
- BIP2152I: < IB10NODE.server1 > Configuration message received from broker. [10/3/2014 5:07:36 PM]
- BIP2153I: < IB10NODE.server1 > About to "Change" an execution group. [10/3/2014 5:07:36 PM]
- BIP2155I: < IB10NODE.server1 > About to "create" the deployed resource "EmployeeService_JSONClient" of type ".APPZIP". [10/3/2014 5:07:37 PM]
- BIP2155I: < IB10NODE.server1 > About to "create" the deployed resource "gen.getEmployee_EmployeeService_EmpServClient_JSON1" of type ".SUBFLOW". [10/3/2014 5:07:37 PM]
- BIP2155I: < IB10NODE.server1 > About to "create" the deployed resource "EmpServClient_JSON1" of type ".MSGFLOW". [10/3/2014 5:07:37 PM]
- BIP2154I: < IB10NODE.server1 > Execution group finished with Configuration message. [10/3/2014 5:07:43 PM]
- BIP3132I: < IB10NODE.HTTPListener > The HTTP Listener has started listening on port '7080' for 'http' connections. [10/3/2014 5:07:47 PM]
- BIP2152I: < IB10NODE.server1 > Configuration message received from broker. [10/3/2014 5:50:41 PM]
- BIP2153I: < IB10NODE.server1 > About to "Change" an execution group. [10/3/2014 5:50:41 PM]
- BIP2155I: < IB10NODE.server1 > About to "delete" the deployed resource "EmpS

This tool is not shipped as part of the IIB product; please contact us directly if you would like a copy.

7.3 Configure TESTNODE_iibuser for REST APIs

The instructions in this lab guide are based on a Windows implementation, with a user named "iibuser". Login to Windows as the user "iibuser", password = "passw0rd". (You may already be logged in). Start the IIB Toolkit from the Start menu.

The IIB support for the REST API requires some special configuration for the IIB node and server. Cross-Origin Resource Scripting (CORS) must be enabled for the IIB node to execute REST applications. This is also required when testing with the SwaggerUI test tool. See http://www.w3.org/TR/cors/?cm_mc_uid=09173639950214518562833&cm_mc_sid_50200000=1452177651 for further information.

5. Ensure that TESTNODE_iibuser is started.
6. Check that CORS has been enabled on the IIB node by running the following command in an Integration Console:

```
mqswirereportproperties TESTNODE_iibuser
-e default
-o HTTPConnector
-r
```

7. If CORS is enabled, you will see the following lines (amongst others):

```
corsEnabled='true'
corsAllowOrigins='*'
corsAllowCredentials='false'
corsExposeHeaders='Content-Type'
corsMaxAge='-1'
corsAllowMethods='GET, HEAD, POST, PUT, PATCH, DELETE, OPTIONS'
corsAllowHeaders='Accept, Accept-Language, Content-Language, Content-Type'
```

8. If CORS has not been enabled, run the following commands:

```
mqsicchangeproperties TESTNODE_iibuser
-e default
-o HTTPConnector
-n corsEnabled -v true

mqsistop TESTNODE_iibuser

mqsistart TESTNODE_iibuser
```

8. Create the HRDB database in MongoDB

This chapter will provide some very basic instructions to get you started with MongoDB, sufficient to perform this lab. There are many good instructional documents on the internet, starting with this site: <https://www.mongodb.org/>.

Note - on the IIB Workshop Windows system, MongoDB has been installed into the Program Files folder, and c:\Program Files\MongoDB\server\3.2\bin has been added to the PATH in the Windows System Environment Variables.

8.1 Start the MongoDB server and client shell

1. In a Windows Command Prompt, navigate to :

```
c:\student10\Loopback\mongodb\commands
```

Run the command:

```
startMongoDB
```

For info, this will run the MongoDB command:

```
mongod.exe --dbpath c:\student10\Loopback\mongodb\data\db
```

This command will start the MongoDB server, and create any new databases in the specified folder. The folder specified in the dbpath parameter must exist, but does not need any specific preparation or configuration.

No defaults have been changed, so the MongoDB server will start with the client listener 27017. The DOS window will be held open at this point. Do not close this window, or the MongoDB server will terminate.

```
C:\student10\Loopback\mongodb\commands>startMongoDB.cmd
C:\student10\Loopback\mongodb\commands>mongod --dbpath c:\student10\loopback\mongodb\data\db
2016-05-03T08:32:49.109+0100 I CONTROL [main] Hotfix KB2731284 or later update
is not installed, will zero-out data files
2016-05-03T08:32:49.114+0100 I CONTROL [initandlisten] MongoDB starting : pid=9
92 port=27017 dbpath=c:\student10\loopback\mongodb\data\db 64-bit host=BETAWORKS
-ESB10
2016-05-03T08:32:49.118+0100 I CONTROL [initandlisten] targetMinOS: Windows 7/W
indows Server 2008 R2
2016-05-03T08:32:49.120+0100 I CONTROL [initandlisten] db version v3.2.5
2016-05-03T08:32:49.120+0100 I CONTROL [initandlisten] git version: 34e65e5383f
7ea1726332cb175b73077ec4a1b02
2016-05-03T08:32:49.122+0100 I CONTROL [initandlisten] OpenSSL version: OpenSSL
1.0.1p-fips 9 Jul 2015
2016-05-03T08:32:49.125+0100 I CONTROL [initandlisten] allocator: tcmalloc
2016-05-03T08:32:49.126+0100 I CONTROL [initandlisten] modules: none
2016-05-03T08:32:49.128+0100 I CONTROL [initandlisten] build environment:
2016-05-03T08:32:49.131+0100 I CONTROL [initandlisten] distmod: 2008plus-ss
1
2016-05-03T08:32:49.133+0100 I CONTROL [initandlisten] distarch: x86_64
2016-05-03T08:32:49.135+0100 I CONTROL [initandlisten] target_arch: x86_64
2016-05-03T08:32:49.136+0100 I CONTROL [initandlisten] options: { storage: { db
Path: "c:\student10\loopback\mongodb\data\db" } }
```

2. Start a Mongo client shell.

Open a new Windows Command Prompt, and execute the command "mongo".

This will use the default port of 27017, and connect to the started server.

Note that the mongo client will initially connect to the server, and will connect to the **test** database.

```
C:\Users\iibuser>mongo
2016-05-03T09:39:58.417+0100 I CONTROL  [main] Hotfix KB2731284 or later update is not installed, will zero-out data files
MongoDB shell version: 3.2.5
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
    http://docs.mongodb.org/
Questions? Try the support group
    http://groups.google.com/group/mongodb-user
Server has startup warnings:
2016-05-03T09:39:50.790+0100 I CONTROL  [initandlisten]
2016-05-03T09:39:50.790+0100 I CONTROL  [initandlisten] ** WARNING: Insecure configuration, access control is not enabled and no --bind_ip has been specified.
2016-05-03T09:39:50.791+0100 I CONTROL  [initandlisten] ** Read and write access to data and configuration is unrestricted.
2016-05-03T09:39:50.792+0100 I CONTROL  [initandlisten] ** and the server listens on all available network interfaces.
2016-05-03T09:39:50.793+0100 I CONTROL  [initandlisten]
>
```

8.2 Create the HRDB database

With MongoDB, it is not necessary to explicitly define a database. A database is created on first reference, so you will do this now.

3. As mentioned above, the default database connection is "test", so switch to the required database name by issuing the mongo shell command :

```
use HRDB
```

```
2018-05-03T07:57:58.173
> use HRDB
switched to db HRDB
> -
```

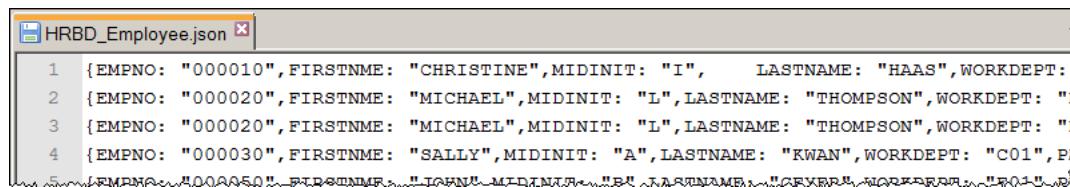
4. The lab has provided a set of data for the HRDB database. In Windows Explorer, locate the file

```
c:\student10\Loopback\mongodb\createDB\
HRDB_Employee_batch_load.json
```

Open this file with Notepad++.

You will see the familiar contents of the HRDB EMPLOYEE data in JSON format.

Each line has a "{" initiator and "}" terminator.



Close the file without saving any changes.

5. You will now import this json data into a "Collection" called **EMPLOYEE** in the **HRDB** database.

In a new Windows command prompt (not the window running the mongo shell), navigate to:

```
c:\student10\Loopback\mongodb\createDB\
```

Enter the command:

```
mongoimport --db HRDB
--collection EMPLOYEE
--drop
--file HRDB_Employee_batch_load.json
```

This command will create a collection called **EMPLOYEE** and import the json data in **HRDB_Employee_batch_load.json** (deleting any previous collection).

6. The mongo import command will respond similar to the following:

```
2016-05-12T16:34:12.710+0100      connected to: localhost
2016-05-12T16:34:12.714+0100      dropping: HRDB.EMPLOYEE
2016-05-12T16:34:12.742+0100      imported 44 documents
```

7. Check that you can retrieve an entry from the Collection.

In the window running the Mongo client shell, execute the command:

```
db.EMPLOYEE.find ( { EMPNO: "000010" } )
```

This should return a single document containing the details of employee 000010;.

```
> db.EMPLOYEE.find({EMPNO: "000010"})
{ "_id" : ObjectId("57347fce015454d44dea9d45"), "EMPNO" : "000010", "FIRSTNAME" :
"CHRISTINE", "MIDINIT" : "I", "LASTNAME" : "HAAS", "WORKDEPT" : "A00", "PHONE" :
"3978", "HIREDATE" : "1995-01-01", "JOB" : "PRES", "EDLEVEL" : 18, "SEX" : "F",
"BIRTHDATE" : "1963-08-24", "SALARY" : 152750, "BONUS" : 1000, "COMM" : 4220 }
```

8. Finally, you can select all documents, by using the command

```
db.EMPLOYEE.find ()
```

```
> db.EMPLOYEE.find()
{ "_id" : ObjectId("5734a274015454d44dea9d70"), "EMPNO" : "000020", "FIRSTNAME" :
MICHAEL", "MIDINIT" : "L", "LASTNAME" : "THOMPSON", "WORKDEPT" : "B01", "PHONE" :
"3476", "HIREDATE" : "2003-10-10", "JOB" : "MANAGER", "EDLEVEL" : 18, "SEX" : "M",
"BIRTHDATE" : "1978-02-02", "SALARY" : 94250, "BONUS" : 800, "COMM" : 3300 }
{ "_id" : ObjectId("5734a274015454d44dea9d71"), "EMPNO" : "000010", "FIRSTNAME" :
"CHRISTINE", "MIDINIT" : "I", "LASTNAME" : "HAAS", "WORKDEPT" : "A00", "PHONE" :
"3978", "HIREDATE" : "1995-01-01", "JOB" : "PRES", "EDLEVEL" : 18, "SEX" : "F",
"BIRTHDATE" : "1963-08-24", "SALARY" : 152750, "BONUS" : 1000, "COMM" : 4220 }
{ "_id" : ObjectId("5734a274015454d44dea9d72"), "EMPNO" : "000020", "FIRSTNAME" :
"MICHAEL", "MIDINIT" : "L", "LASTNAME" : "THOMPSON", "WORKDEPT" : "B01", "PHONE" :
"3476", "HIREDATE" : "2003-10-10"..... etc
```

9. Now that you have loaded some test data into the MongoDB database HRDB, the next chapter will show you how to access this from IIB.

9. Configure the IIB installation

You can use a LoopBackRequest node in a message flow to access or create records through a LoopBack connector. First, you must install your chosen connector to work with IBM Integration Bus, and then configure the data source for the connector. You can then specify the security credentials that are required to access it. The following topics describe the steps involved in completing those tasks:

1. Installing the LoopBack connector
2. Configuring the data source and models for your LoopBack connector
3. Optionally - specifying security credentials for connecting to a secured data source

9.1 Install the LoopBack Connector

12. If you happened to run the preceding Kafka lab, then close any command windows which may be open. Then, open an Integration Bus Command Console, change directory to the node_modules folder of the work path directory (defined by the MQSI_WORKPATH environment variable):

```
cd \
cd %MQSI_WORKPATH%\node_modules
```

Install the required loopback connector for MongoDB by running the command

```
npm install loopback-connector-mongodb
```

This will create the folder **node-modules**, which contains the loopback connector for mongodb. ("npm" is a nodejs command, which is installed on the workshop Windows system.)

The output of the command should look like this. You can use Windows Explorer to check the contents of the connectors\node_modules folder.

```
C:\IBMMIIB\10.0.0.6>cd\
C:\>cd %MQSI_WORKPATH%\node_modules
C:\ProgramData\IBM\MQSI\node_modules>npm install loopback-connector-mongodb
loopback-connector-mongodb@1.15.2 loopback-connector-mongodb
├── async@1.5.2
├── debug@2.2.0 <ms@0.7.1>
└── loopback-connector@2.4.0
    mongodb@2.2.9 <es6-promise@3.2.1, readable-stream@2.1.5, mongodb-core@2.0.11>

C:\ProgramData\IBM\MQSI\node_modules>
```

9.2 Configure data source for Loopback connector

When you have installed a LoopBack connector to be used with IBM Integration Bus, you must configure the data source for the connector, by adding a connector-specific data-source stanza to the datasources.json file.

1. The creation process for the loopback connector does not automatically create the datasources.json connections details, so this must be created manually. **Depending on your VM image this may already have been done for you, so read through these steps just to make sure ...**

In this example, we have provided the required datasources file, so using Windows Explorer, copy the file c:\student10\loopback\mongodb\config\datasources.json.

Paste this file into the folder into the **MQSI_WORKPATH\connectors\loopback** folder. On the provided Windows workshop system, this folder will be **c:\ProgramData\IBM\MQSI\connectors\loopback**.

Using Notepad++, open this file and take a look at the contents. The required stanza is the “mongodb” stanza, which corresponds to the data source name in the Loopback node properties in the IIB Toolkit. If you are using your own installation, you will need to change the hostname and port values accordingly.

```
datasources.json
1 [
2   "memory": {
3     "name": "memory",
4     "connector": "memory",
5     "file": "c:/student10/loopback/mongodb/memory/mydata.json"
6   },
7   "mongodb": {
8     "host": "betaworks-esb10",
9     "port": 27017,
10    "database": "HRDB",
11    "name": "mongodb",
12    "connector": "mongodb"
13  }
14 }
15 }
```

10. Create an IIB REST API to access MongoDB

10.1 Create the new REST API

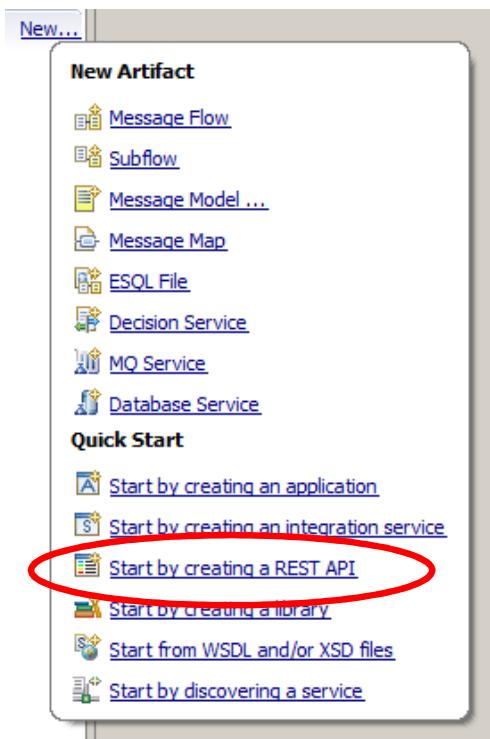
1. In the IIB Toolkit, create a new workspace called

```
c:\user\iibuser\IBM\IIBT10\workspace_loopback
```

Use File, Switch workspace, Other to do this.

Note that it is important to restart the IIB Toolkit at this point, and to create a new workspace. The changes that you made to the MQSI_WORKPATH in section 3 can only be picked up by the IIB Toolkit after a restart.

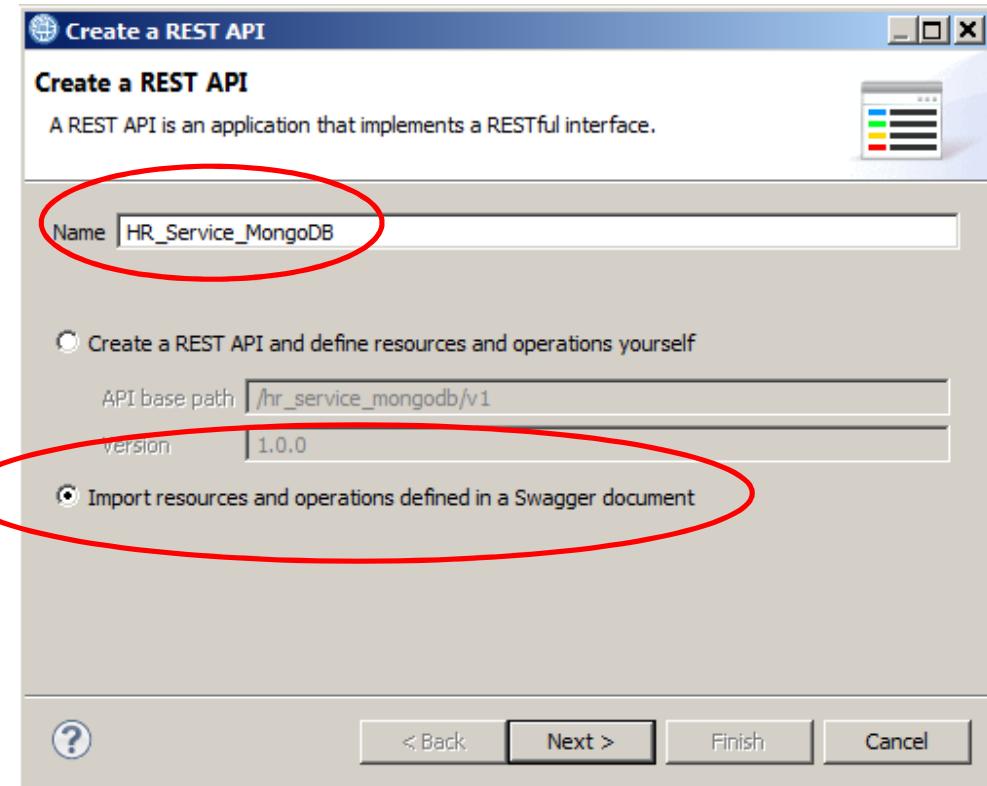
2. In the workspace, create a new REST API..



3. Name the new service HR_Service_MongoDB

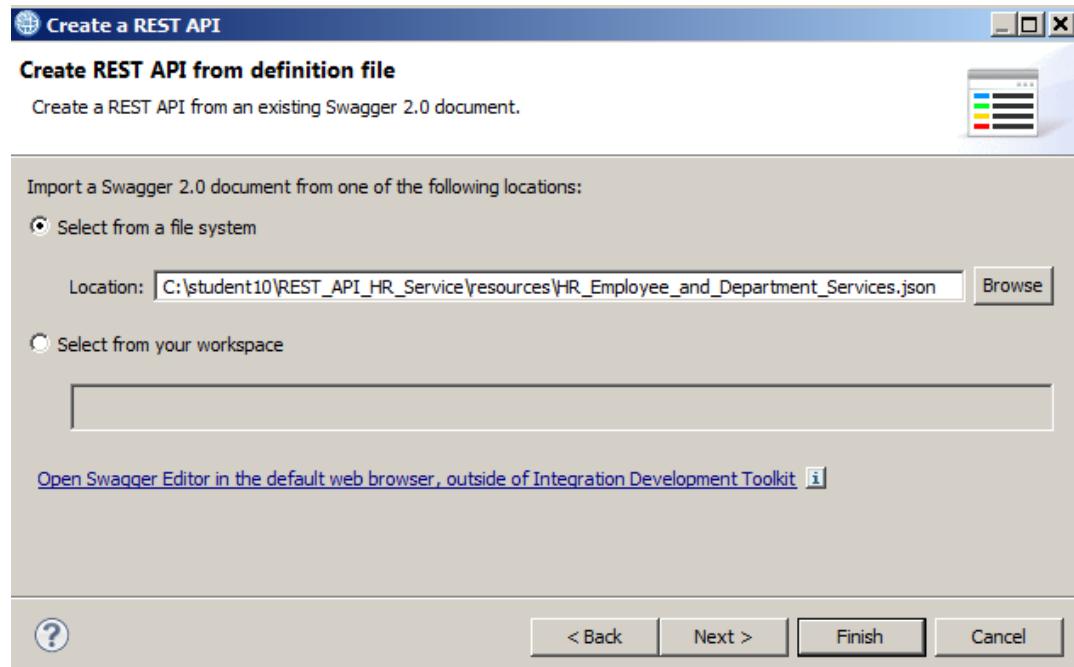
Select "Import resources and operations defined in a Swagger document".

Click Next.



4. Using the Browse button, import the JSON document

```
c:\student10\REST_API_HR_Service\resources\  
HR_Employee_and_Department_Services.json
```



5. The summary window will show you all of the REST operations that were defined in the JSON document. These operations were constructed to match the EMPLOYEE and DEPARTMENT tables in the HRDB database.

Note there is an operation named getEmployees (ie. retrieve a list of all employees), and an operation named getEmployee. This lab will implement the **getEmployee** operation.

Click Finish.

The screenshot shows the 'Create a REST API' interface with the title 'Review the imported REST API definition.' The window displays the following information:

- Title:** HR Employee and Department Services
- Description:** This is the HR Swagger document for the Employee and Department Services used by the IIB BetaWorks REST labs. It contains resource definitions and JSON model definitions.
- Base path:** /HR_Services/resources
- Version:** 3.0.0

Operation	Method	Resource
getDepartments	GET	/departments
createDepartment	POST	/departments
getDepartment	GET	/departments/{departmentKey}
updateDepartment	PUT	/departments/{departmentKey}
deleteDepartment	DELETE	/departments/{departmentKey}
getDepartmentEmploy...	GET	/departments/{departmentKey}/employees
getDepartmentManager	GET	/departments/{departmentKey}/manager
setDepartmentManager	PUT	/departments/{departmentKey}/manager
getEmployees	GET	/employees
createEmployee	POST	/employees
getEmployee	GET	/employees/{employeeNumber}
updateEmployee	PUT	/employees/{employeeNumber}
deleteEmployee	DELETE	/employees/{employeeNumber}
getEmployeeDepartment	GET	/employees/{employeeNumber}/department
setEmployeeDepartment	PUT	/employees/{employeeNumber}/department
predictSalary	GET	/employees/{EDLEVEL}/predictSalary

At the bottom of the window are buttons for '?', '< Back' (disabled), 'Next >', 'Finish', and 'Cancel'.

6. The swagger document has now been imported into the Integration Toolkit. The import process has also created a base REST application and a message flow that implements the REST API.

The imported and generated items are split into five main sections in the REST API editor:

- Header - containing the base URL for the REST API, title and description
- Resources - containing all the resources from the swagger document, and all of the operations that are contained within each resource
- Model Definitions - schema definitions for the input and output JSON objects
- Error Handling - options to add some elements of runtime security
- Security - basic security parameters

HR_Service

- ▶ Header
- ▼ Resources
 - ▶ /departments
 - ▶ /departments/{departmentKey}
 - ▶ /departments/{departmentKey}/employees
 - ▶ /departments/{departmentKey}/manager
 - ▶ /employees
 - ▶ /employees/{EDLEVEL}/predictSalary
 - ▶ /employees/{employeeNumber}
 - ▶ /employees/{employeeNumber}/department
- ▼ Model Definitions

Name
(+) <Enter a unique name to create a new model>
(+) {...} EMPLOYEE
(+) {...} DEPARTMENT
(+) {...} DBRESP
(+) {...} EmployeeResponse
(+) {...} DepartmentResponse
(+) {...} DetailedResponse

7. As an example of Resources, expand **/employees/{employeeNumber}**. (You may wish to collapse the **/department** resource, for readability).

You will see three operations, GET, PUT and DELETE.

For some of the operations (for example, the updateEmployee PUT operation in this resource), the Schema type has been set (in this case to EMPLOYEE).

For some other operations (for example the getEmployee GET operation), the input parameter is specified (EMPNO). The Schema Type of the successful (200) operation has been set to EmployeeResponse (originally specified in the swagger doc).

You can use the Schema Type dropdowns to change the required schema for the operation. The available values are derived from the Model Definitions section. If no schema type is specified, the REST operation can dynamically specify the format of the output message.

The screenshot shows the IBM Integration Bus Swagger UI interface for the **/employees/{employeeNumber}** resource. It displays three operations: **getEmployee**, **updateEmployee**, and **deleteEmployee**.

- getEmployee:** A GET operation to retrieve employee details. The **Schema type** dropdown for the 200 OK response is circled in red, showing it is set to **EmployeeResponse**.
- updateEmployee:** A PUT operation to update an existing employee. The **Schema type** dropdown for the Request body is circled in red, showing it is set to **EMPLOYEE**.
- deleteEmployee:** A DELETE operation to delete an existing employee.

The interface includes tables for parameters, responses, and descriptions, with columns for Name, Parameter type, Data type, Format, Required, Description, Response status, Description, Array, Schema type, and Allow null.

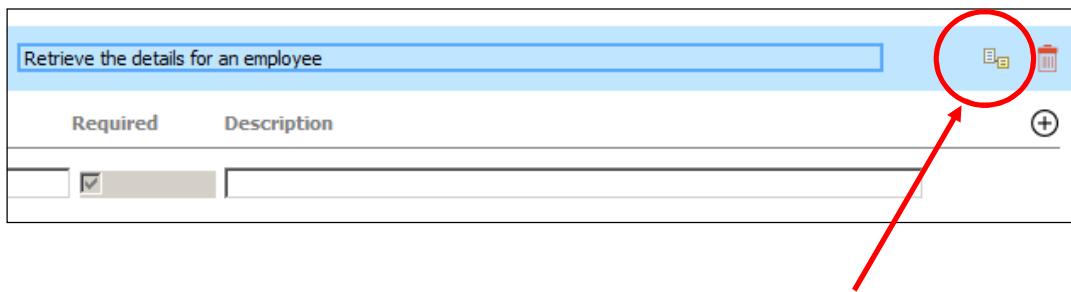
8. Expand the REST API Resources, and position the editor at the **/employees/{employeeNumber}** resource.

The getEmployee operation is the first operation in this resource.

Name	Parameter type	Data type	Format	Required	Description
employeeNumber	path	string		<input checked="" type="checkbox"/>	

Response status	Description	Array	Schema type
200	OK	<input type="checkbox"/>	EmployeeResponse
500	Something wrong in Server	<input type="checkbox"/>	
404	The employee cannot be found	<input type="checkbox"/>	

9. In the top right part of the description, click on the icon to "Create a subflow for the operation".



This will create a skeleton subflow where you will provide the logic to implement the operation:



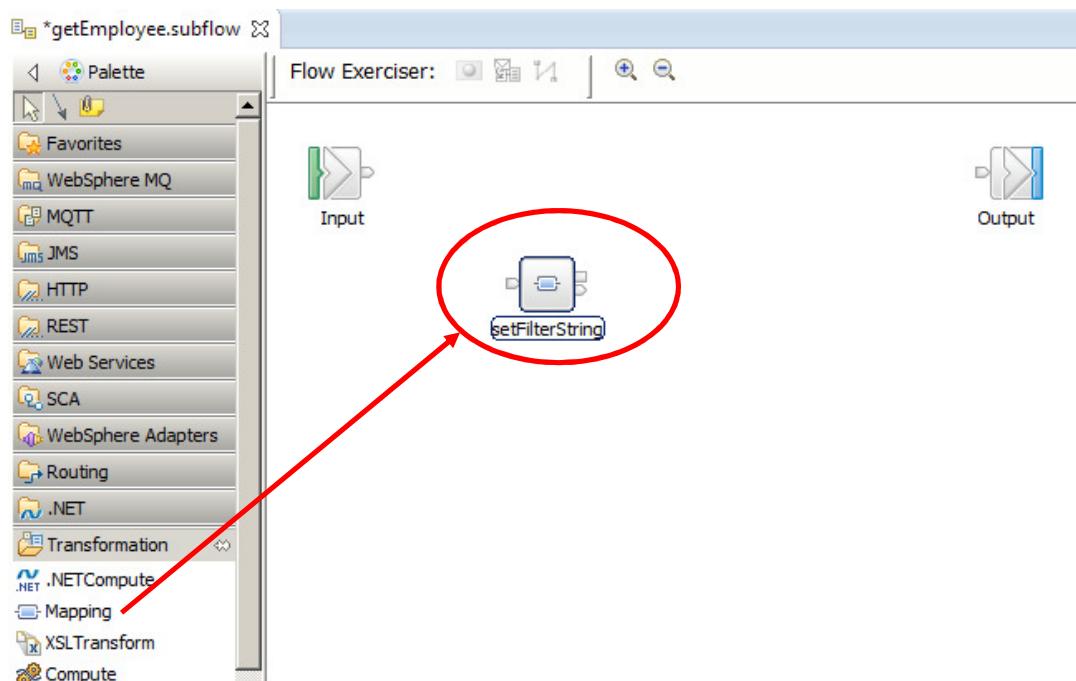
Two alternatives are provided in this lab:

1. In the first alternative, you will use a new Mapping node to set the value of the LocalEnvironment filterString element that is used by the Loopback Request node.
2. In the second alternative, you will use an ESQL Compute node to set the value of the filterString element.

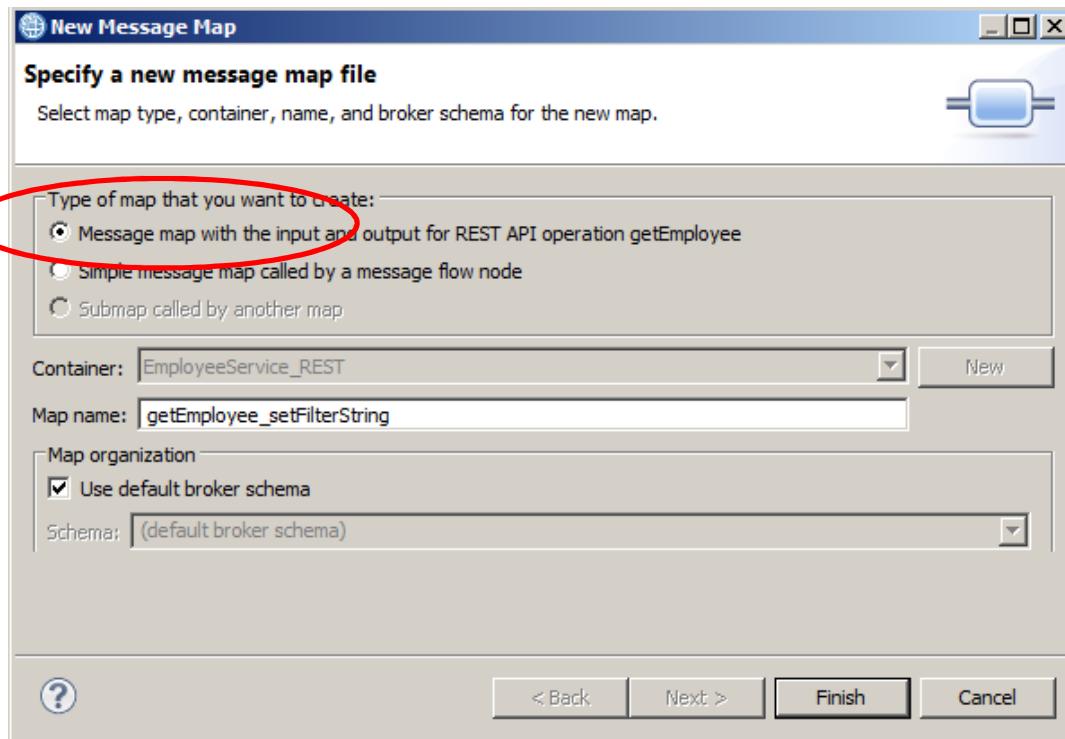
If you wish to use ESQL, skip to "10.3 Alternative 2 - using ESQL".

10.2 Alternative 1: Set LocalEnvironment "filterString" using Mapping Node

1. Add a new Mapping node to the flow editor; call it setFilterString.



2. Open the new map. At the New Message Map window, accept the default (Message Map for REST API operation getEmployee), and click Finish.



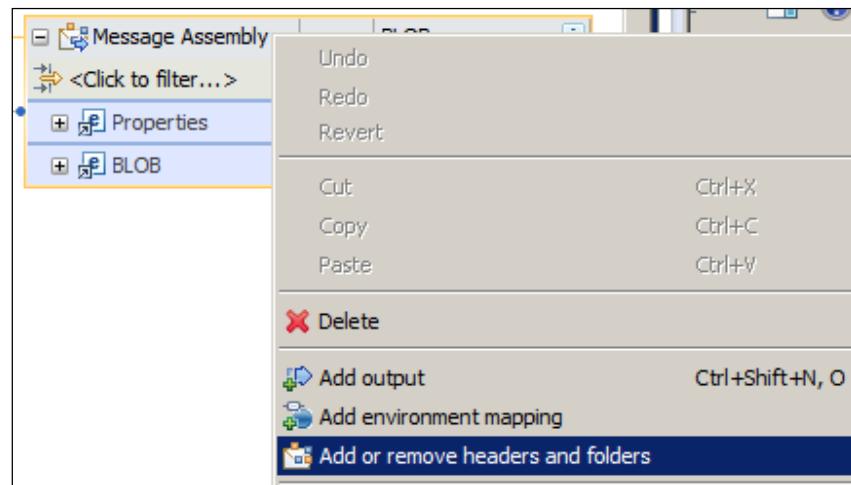
3. Note that LocalEnvironment has already been added to the input message assembly.

Expanding this, and the REST folder, will show the input parameter employeeNumber. You will use this when setting the value of the output **filterString** parameter for the MongoDB access.

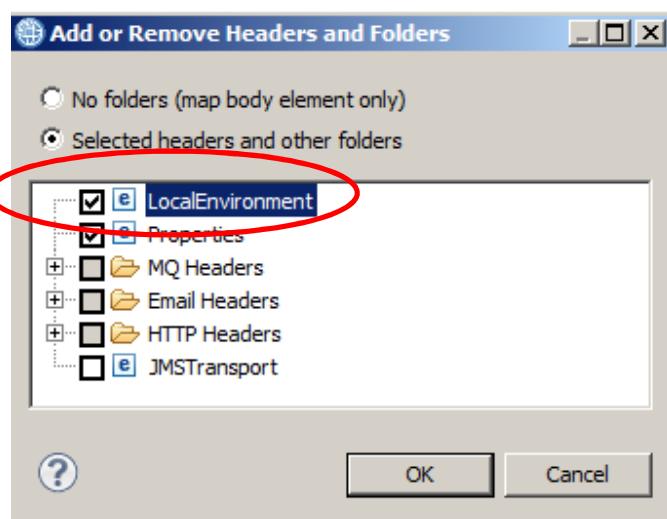
REST	[0..1]	_LocalEnvironmentRESTType
Input	[0..1]	_RESTInputType
Method	[0..1]	string
Operation	[0..1]	string
Path	[0..1]	string
URI	[0..1]	string
Parameters	[0..1]	<Anonymous>
choice of cast items	[0..*]	
any	[1..1]	
employeeNumber	[1..1]	string
Response	[0..1]	_RESTResponseType

4. On the output message assembly, the LocalEnvironment has not been added, so add it now.

Right-click the Message Assembly title, and select "Add or remove headers and folders".



5. Select LocalEnvironment, then click OK.

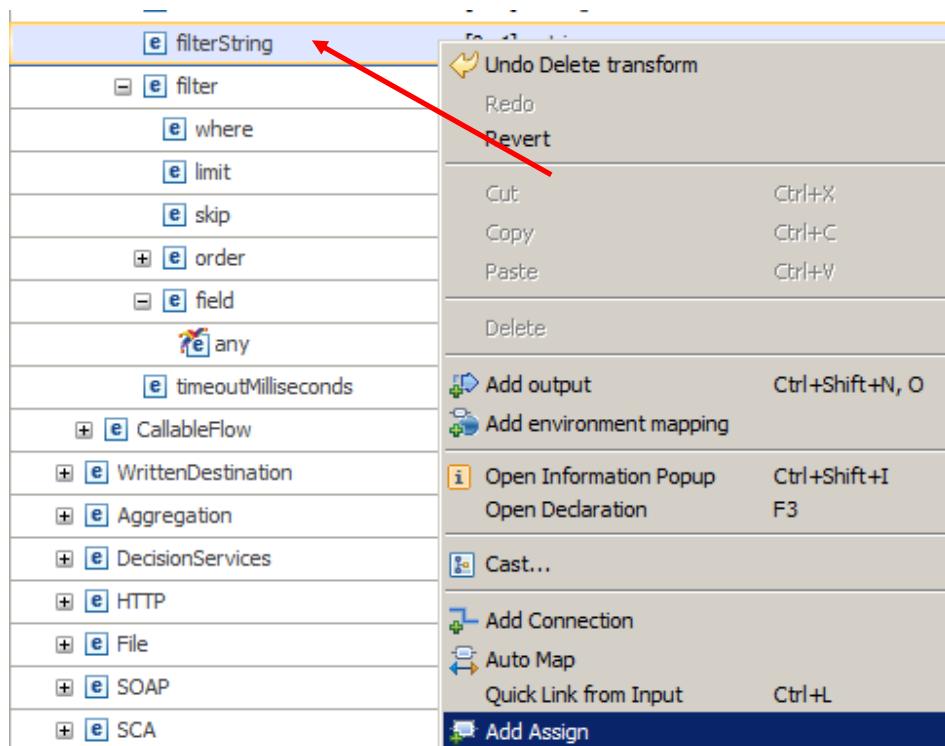


6. In the output LocalEnvironment, fully expand the **Destination/Loopback/Request** folder.

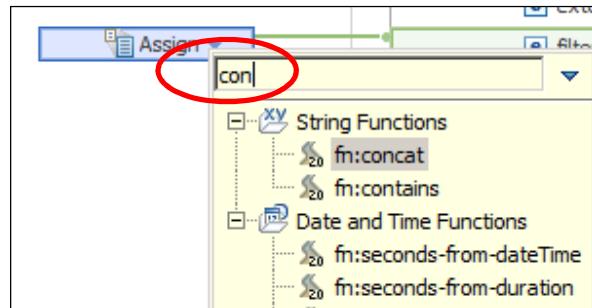
You will see several elements. For this scenario, you will use the filterString element.

[+] e Loopback	[0..1] _LoopbackDestinationType
[+] e Request	[0..1] _LoopbackRequestType
e operation	[0..1] string
e object	[0..1] string
e id	[0..1] string
e externalIdName	[0..1] string
e externalId	[0..1] string
e filterString	[0..1] string
[+] e filter	[0..1] _LoopbackRequestFilterType
e timeoutMilliseconds	[0..1] positiveInteger

7. Right-click filterString, and select "Add Assign" from the context menu.

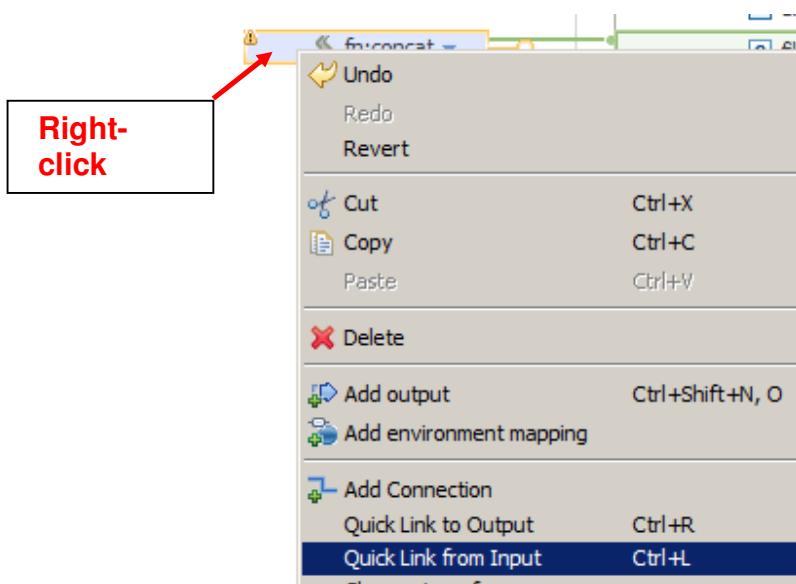


- Click the blue drop-down arrow on the Assign transform, and using the search field, change the transform to a "fn:concat".



- The Concat transform needs to take input from the EMPNO input element, so a connection to this has to be provided.

Right-click the "fn:concat" transform and select "Quick link from Input".



10. In the yellow pop-up window, collapse the Properties folder, expand the LocalEnvironment folder, and locate the REST/Input/Parameters folder.

Select (click) the employeeNumber element.

The screenshot shows a hierarchical tree structure within a yellow-highlighted area. The root node is 'LocalEnvironment'. Under 'LocalEnvironment', there are several nodes: 'JMS', 'Mapping', 'Database', 'MQ', 'MQTT', 'REST', 'Input', 'Method', 'Operation', 'Path', 'URI', 'Parameters', 'any', and 'employeeNumber'. The 'employeeNumber' node is highlighted with a blue selection box. A status bar at the bottom of the window says 'Select input object to complete transformation'.

11. The properties of the fn:concat transform will initially show two parameters (on the General tab). Using the Add button, add a third one, as shown:

The screenshot shows the 'Transform - concat' dialog box. On the left, there is a navigation pane with tabs: General (selected), Variables, Condition, Sort, Order, and Documentation. The 'General' tab contains a 'Description' field with the text: 'Takes any number of arguments, converts them to strings and concatenates them, returning a single string. For example, fn:concat("Happy", " birthday, ", "friend", "!") returns "Happy birthday, friend!" See [XPath 2.0 Specification...](#)'. Below this is a 'Parameters:' section with a table:

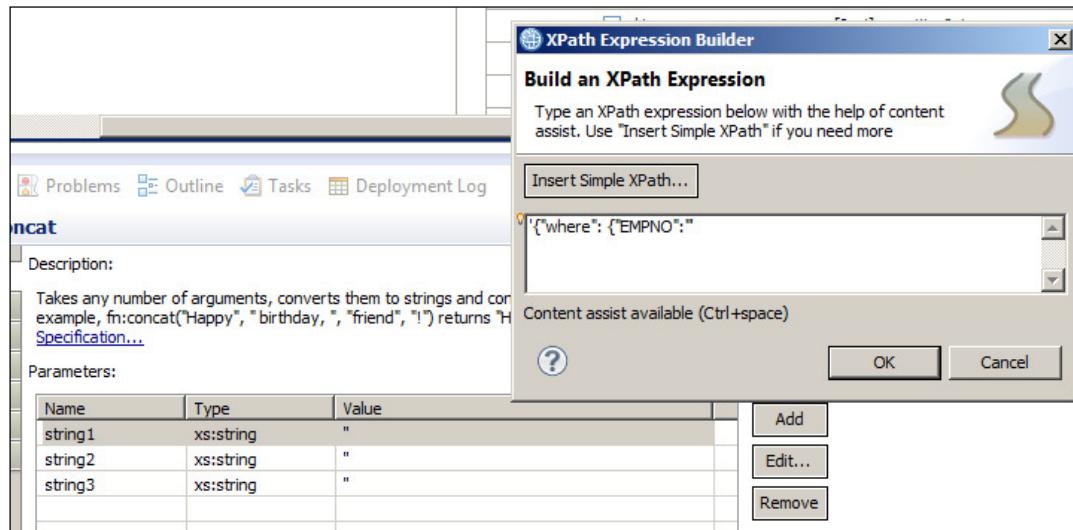
Name	Type	Value
string1	xs:string	"
string2	xs:string	"
string3	xs:string	"

To the right of the table are three buttons: 'Add...', 'Edit...', and 'Remove'. The 'Add...' button is circled in red.

12. Highlight the first parameter, string1, and using the Edit button, set the value of this parameter to

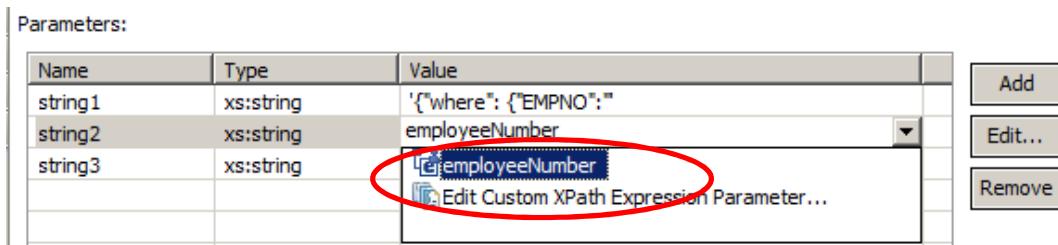
```
'{"where": {"EMPNO": "}'
```

Click OK.



13. Set the value of string2 to \$employeeNumber, by using the drop-down arrow that will be shown when you select this element.

Click here:



Result:

Parameters:		
Name	Type	Value
string1	xs:string	'{"where": {"EMPNO": "'
string2	xs:string	\$employeeNumber
string3	xs:string	"

14. Finally, set the value of string3 to

' " } } '

Parameters:

Name	Type	Value	
string1	xs:string	'{"where": {"EMPNO": "	
string2	xs:string	\$employeeNumber	
string3	xs:string	"}}'	

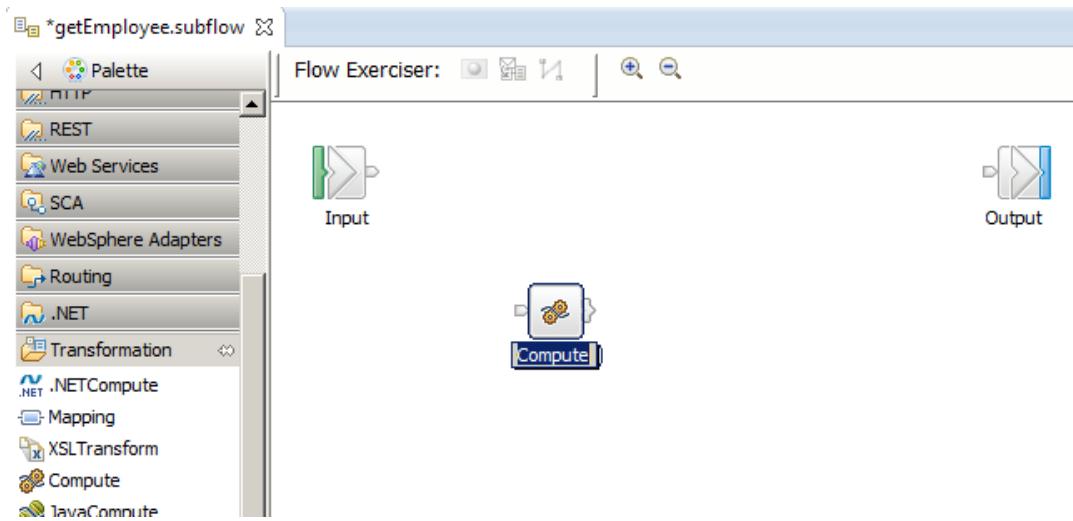
Add
Edit...
Remove

Save and close the map.

15. Skip to 10.4 Complete the Flow Development.

10.3 Alternative 2: Set LocalEnvironment "filterString" using ESQL Compute Node

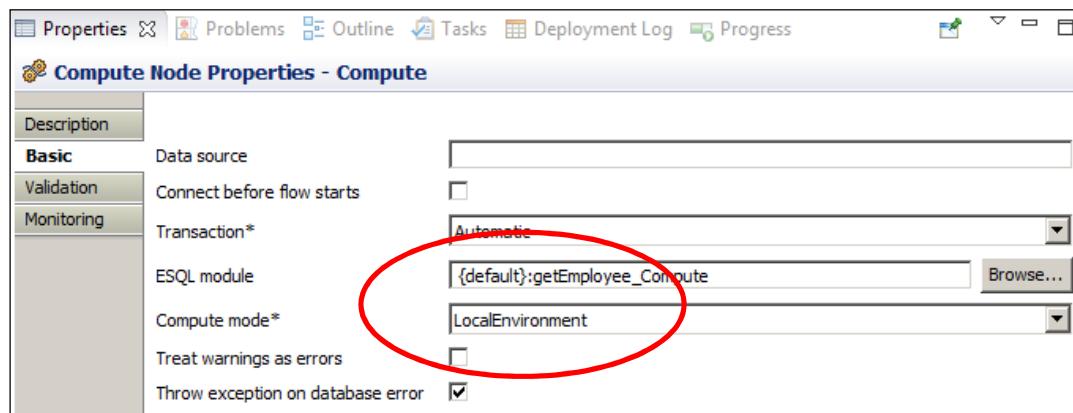
1. Add a new Compute (ESQL) node to the flow editor.



2. Select the Compute node, and in the node properties, set Compute mode to **"LocalEnvironment"**.

This is required because the ESQL in the Compute node will set the value of an element in the Loopback folder of the LocalEnvironment. This needs to be propagated to the Loopback Request node, so that it can be used to pass the query to the MongoDB database.

You can, of course, set the Compute Mode to other values, providing that the LocalEnvironment is included.



3. Open the Compute node, and add a new line of ESQL after the comment line, as shown:

```
-- CALL CopyEntireMessage();  
  
set OutputLocalEnvironment.Destination.Loopback.Request.filterString =  
    '{"where": {"EMPNO": "'  
    || InputLocalEnvironment.REST.Input.Parameters.employeeNumber  
    || '"}}';
```

Note, you can copy/paste this line from the file:

```
c:\student10\loopback\mongodb\esql\setFilterString.esql.txt
```

For reference, the ESQL will create a Loopback query element, contained in the **LocalEnvironment/Destination/Loopback/Request/** folder.

As an example, the query will have the following form:

```
filterString = ' {"where": {"EMPNO": "000010"} } '
```

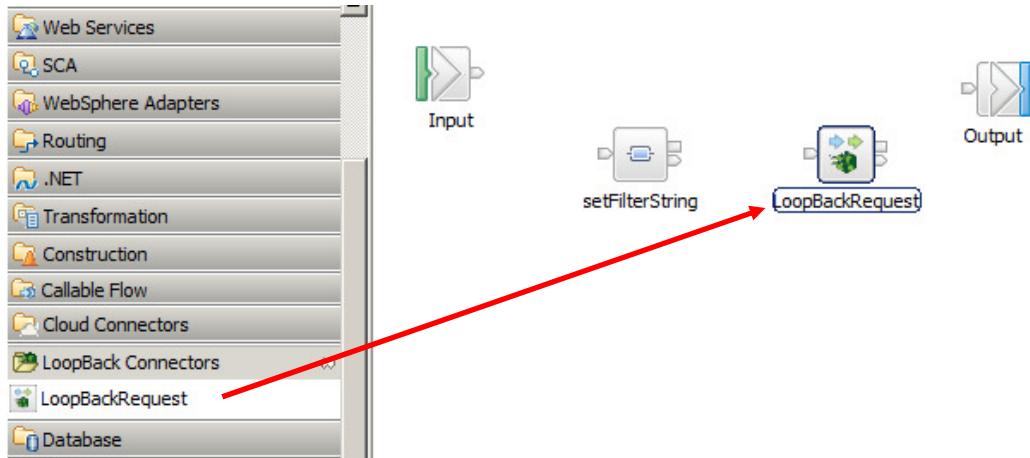
Save and close the ESQL editor.

10.4 Complete the flow development

You will now complete the development of the subflow by using the LoopBack Request node.

The screen captures in this section assume that the Mapping Node has been used to set the filterString, but apply equally if you have used an ESQL Compute node.

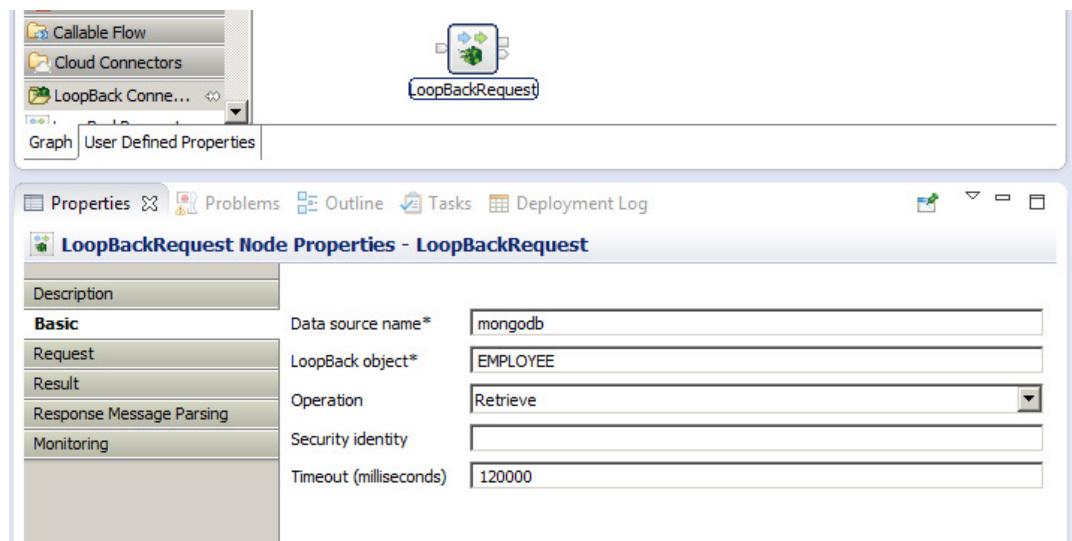
1. From the Loopback Connectors drawer, drop a Loopback Request node onto the flow editor.



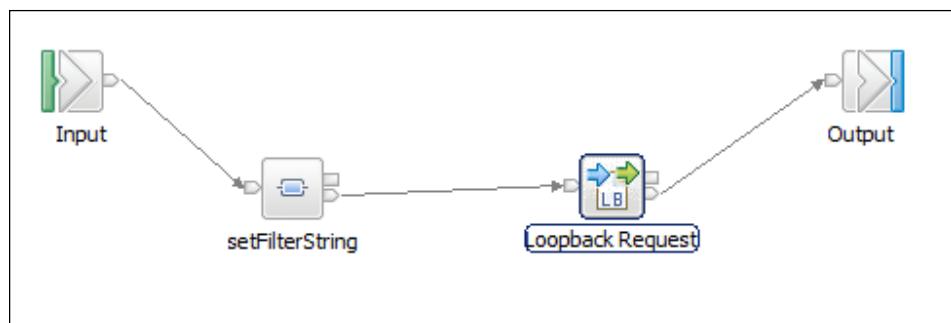
- Select the Loopback Request node, and select the node Properties.

On the Basic tab, set the following values:

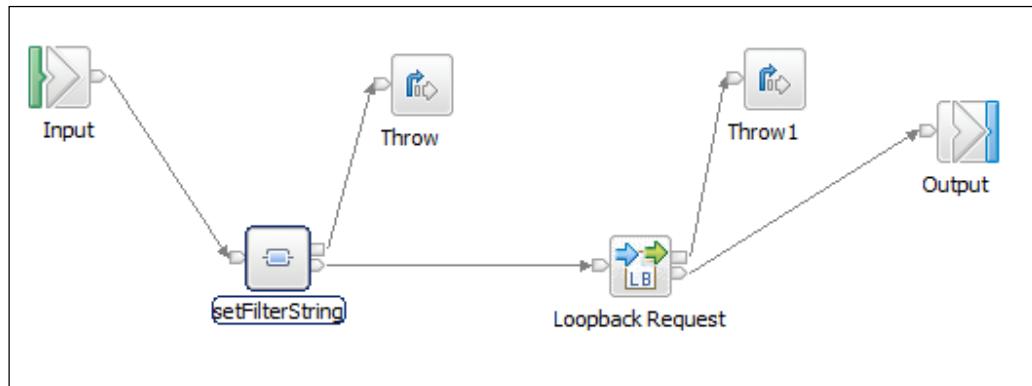
- Name of the data source in the datasources.json file to connect to:
 - mongodb**
- Loopback object
 - EMPLOYEE** (corresponding to the HRDB EMPLOYEE container)
- Operation
 - Retrieve**
- Timeout
 - 1200** (to avoid long timeouts during development ... Note that the screen capture below shows 120000 which should be changed to 1200!)



- Connect the nodes as shown.



4. Connect Throw nodes (Construction folder) to the failure terminals of each of the nodes. Add suitable message text using the properties tab. This will help with any problem determination you may need when you come to test the REST API:



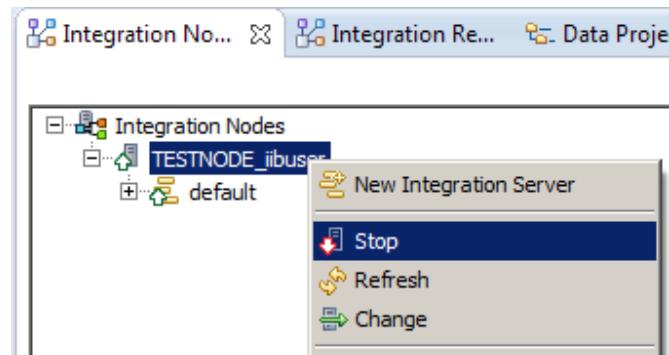
Save the subflow.

10.6 Deploy and test the REST API

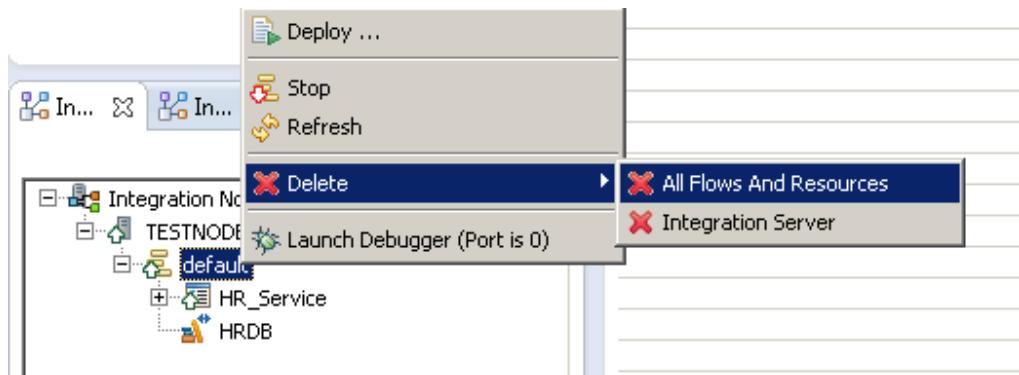
You will now deploy and test the updated REST API. This should retrieve information from the MongoDB HRDB database.

5. First, to activate the changes you made earlier (so that the IIB node can pick up the loopback connector from the MQSI_WORKPATH), stop and restart TESTNODE_iibuser.

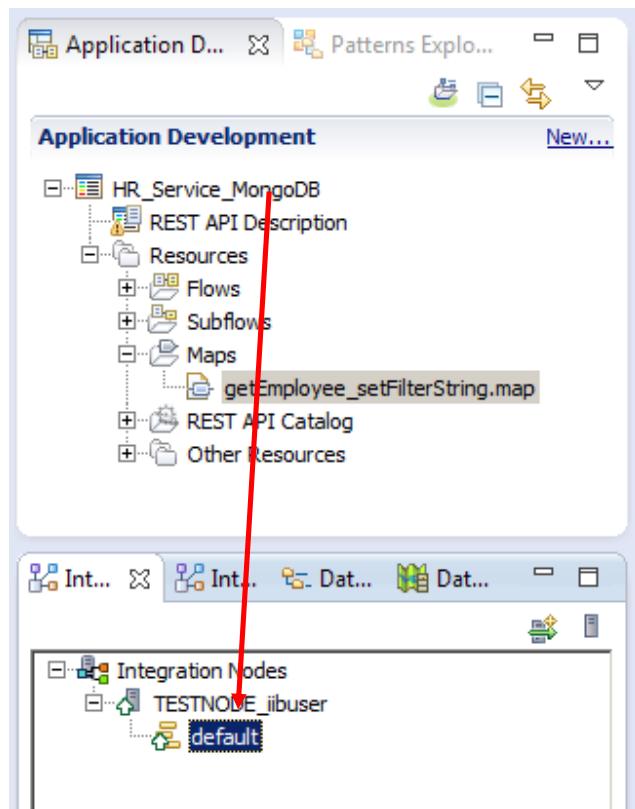
You can do this using the Integration Toolkit (right-click node, Stop, then Start), or a variety of other ways.



6. Ensure all deployed resources are deleted from the default server.

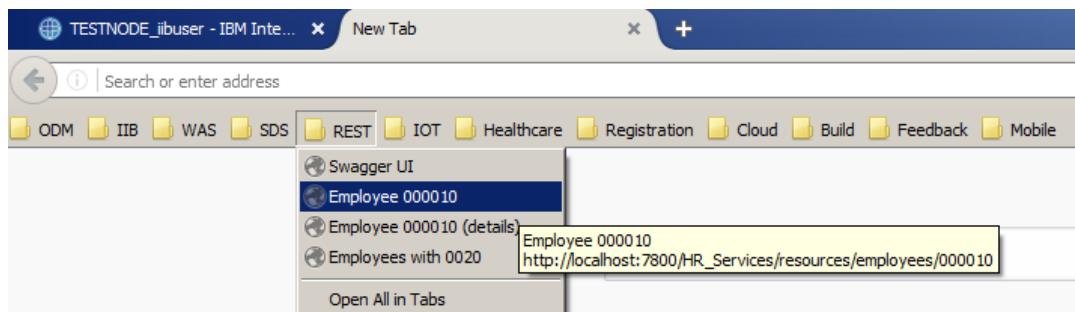


7. In the Integration Toolkit, deploy the HR_Service_MongoDB REST API by dragging and dropping onto the default server.



8. In the supplied Firefox browser, use the shortcut in the REST folder to use the URL:

`http://localhost:7800/HR_Services/resources/employees/000010`



9. This should retrieve the document corresponding the EMPNO = "000010".



A screenshot of a web browser window titled "TESTNODE_ibuser - IBM Inte...". The address bar shows "http://localhost...employees/000010". The page content displays a JSON array with one element:

```
[{"id": "57402e7a0706a97ec5cfe859", "EMPNO": "000010", "FIRSTNAME": "CHRISTINE", "MIDINIT": "I", "LASTNAME": "HAAS", "WORKDEPT": "A00'ES", "EDLEVEL": 18, "SEX": "F", "BIRTHDATE": "1963-08-24", "SALARY": 152750, "BONUS": 1000, "COMM": 4220}]
```

10. Terminate the mongo shell window, and the mongodb server window, using Ctrl-C in both cases.

END OF LOOPBACK LAB SCENARIO